Optimize your applications to the max with Jakarta EE and MicroProfile

Edwin Derks
@edwinderks

09-04-2022
Edwin Derks

- Principal Consultant
- Java Champion
- Writer/Author
- Conference Speaker
- Contributor for Jakarta EE and MicroProfile

@edwinderks

cloudnativesolutions.guru
About this talk

Affinity with Java EE / Jakarta EE and/or Spring (Boot) is recommended

Covers the concepts, tools and tweaks that optimize building and running Jakarta EE / MicroProfile applications from a developer’s perspective

Less code examples / Live coding
Background
Background of Jakarta EE and Spring

1990’s

Virtualize Hardware

CLOUD

now()

Java EE

JAKARTA EE

MicroProfile

by Pivotal

spring boot
Perceptions

🤔 Jakarta EE is **old**!

🤔 Jakarta EE is **slow**!

😢 Jakarta EE consumes lots of **resources**

🚫 Jakarta EE cannot be used for building microservices!
Perceptions Debunked!

- Jakarta EE is **old** feature rich!
- Jakarta EE is **slow** has optimized runtimes!
- Jakarta EE consumes lots of resources
- Jakarta EE cannot be used for building microservices! @see MicroProfile
Optimize Jakarta EE / MicroProfile enterprise apps

BUILD

- Programming Model
- Portability
- Build artifacts
- Developer Experience

RUN

- Application Servers & Runtimes
- Influenced by Java SE

DEPLOY

- Separation of business logic and infrastructure
- Cloud Native Capabilities
Build
Java EE 8 Compatible

Jakarta EE 8

Tooling Release

Jakarta EE 9

New features!

Jakarta EE 10

MicroProfile 5.0

Jakarta EE 9 aligned

MicroProfile 6.0

New features!
Overview

JDK

Java SE

JRE

Runtime

JAR
WAR
EAR

Application Server Runtime

DEPLOYS

Platform

JAKARTA EE

IMPLEMENTS

COMPILERS & BUILDS

RUNS

Maven
Optimize building enterprise applications

JAKARTA EE

<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>9.1.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.eclipse.microprofile</groupId>
  <artifactId>microprofile</artifactId>
  <version>5.0</version>
  <type>pom</type>
</dependency>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-integration</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jersey</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  ....
</dependencies>
### Optimize building enterprise applications

<table>
<thead>
<tr>
<th>BUILD</th>
<th>RUN</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A few Kilobytes</td>
<td>Thin WAR</td>
<td>Fat WAR</td>
</tr>
<tr>
<td>Only your business logic*</td>
<td>Several Megabytes</td>
<td>Your business logic + dependencies</td>
</tr>
<tr>
<td>Often lots of Megabytes</td>
<td>Uber / Fat JAR</td>
<td>Several Kilobytes</td>
</tr>
<tr>
<td>Your business logic + dependencies + runtime</td>
<td>Skimmed JAR</td>
<td>Your business logic + runner</td>
</tr>
</tbody>
</table>

* Your business logic:
  - A few Kilobytes
  - Often lots of Megabytes

**Thin WAR**
- Only your business logic
- A few Kilobytes

**Fat WAR**
- Your business logic + dependencies
- Several Megabytes

**Uber / Fat JAR**
- Your business logic + dependencies + runtime
- Several Kilobytes

**Skimmed JAR**
- Your business logic
- A few Kilobytes

**Deploy**
- BUILD
- RUN
- Deploy
From a security perspective: what's in the box?

- **Thin WAR** and **Skimmed JAR** pose no security threat.

- **Fat WAR** and **Uber JAR** can transitively drag in blacklisted, unwanted, unexpected or conflicting dependencies.

---

**Bert Koorengevel** @BertKoor · Jun 27

Hit the nail on the head...
Adding the Web and Security dependencies in SpringBoot also adds a hand full of known vulnerabilities. One of them is letting it parse an application.yaml file is not entirely safe.
Can look up the CVE numbers for you on Monday.
Run
Application Server

- Runs with Java version supported by the server

<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>9.1.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.eclipse.microprofile</groupId>
  <artifactId>microprofile</artifactId>
  <version>5.0</version>
  <type>pom</type>
</dependency>

- One complete package providing Jakarta EE and/or Microprofile
- What you see is what you get
- Thoroughly tested by vendor
- Easily patched and upgraded
- Enables portability for Thin and Fat WAR
Compatible Application Servers

Jakarta EE 9.1 Platform Compatible Products

- **Apusic AAS**
  - Kingdee Apusic Cloud Computing Co., Ltd.
  - 10-EE9

- **Eclipse GlassFish**
  - Eclipse Foundation
  - 6.1.0RC1

- **InforSuite Application Server**
  - Shandong Cvicse Middleware Co., Ltd.
  - 11

- **ManageFish Server**
  - ManageCat
  - 6.7.0

- **Open Liberty**
  - IBM Corporation
  - 21.0.0.12, Java 17
  - 21.0.0.12, Java 11
  - 21.0.0.12, Java 8

- **Payara Server Community**
  - Payara Services Limited
  - 6.2021.1.Alpha.1

- **TongWeb Application Server**
  - Beijing TongTech Co., Ltd.
  - 9

- **WildFly**
  - Red Hat
  - Preview 25.0.0 Final, Java 17
  - Preview 23.0.2 Final, Java 11
  - Preview 22.0.2 Final, Java 8

https://jakarta.ee/compatibility/
Hollow JAR

- Runs with Java version supported by the runtime
- Application server as one single artifact
- Little to no operational management tools
- Don’t include the whole application server with every build of your application
- “Just enough app server” for Thin and Fat WAR
Uber JAR

- Runs with Java version supported by the runtime
- One single artifact
- Little to no operational management tools
- Contains your code and application server
- Opposite of lean and tidy, so bigger in size
- “Just enough app server”
Skimmed JAR

- Runs with Java version supported by the runtime
- One build artifact, libs are external
- Contains your code and bootstrapper/runner
- Optimized for performance and resource efficiency
- “Just enough app server”, tailored to only contain the components you need
Benefits of choosing a designated JVM

- Like Jakarta EE and MicroProfile, a JVM is a **specification** with various **vendor-specific implementations**

- Just like application servers, each one can run your app but probably behaves differently and has different JVM options available for squeezing out even more benefits

- You are probably already familiar with **Oracle's HotSpot JVM**

- An alternative, general purpose JVM to experiment with is **Eclipse OpenJ9**
Designated JVM + Application Server Tuning

Payara Micro 5.193 HotSpot
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1572613068474] [levelValue: 800] Payara Micro 5.193 #badassmicrofish (build 252) ready in 7.011 (ms)

Payara Micro 5.193 OpenJ9 warm cache + startup tuning
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1572612913003] [levelValue: 800] Payara Micro 5.193 #badassmicrofish (build 252) ready in 2,503 (ms)

Payara Micro 5.193 HotSpot + simple app
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1572613153980] [levelValue: 800] Payara Micro 5.193 #badassmicrofish (build 252) ready in 10.916 (ms)

Payara Micro 5.193 OpenJ9 warm cache + startup tuning + simple app
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1572613205845] [levelValue: 800] Payara Micro 5.193 #badassmicrofish (build 252) ready in 4,726 (ms)

Source: https://blog.openj9.org/2019/10/22/running-payara-micro-on-openj9/amp/?__twitter_impression=true
Designated JVM + Application Server Tuning

Payara Micro 5.2021.10 HotSpot
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1643979258162] [levelValue: 800] Payara Micro  5 #badassmicrofish (build 879) ready in 7,683 (ms)

Payara Micro 5.2021.10 OpenJ9 warm cache + startup tuning
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1643979822119] [levelValue: 800] Payara Micro  5 #badassmicrofish (build 879) ready in 3,853 (ms)

Payara Micro 5.2021.10 HotSpot + simple app
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1643979337194] [levelValue: 800] Payara Micro  5 #badassmicrofish (build 879) ready in 12,925 (ms)

Payara Micro 5.2021.10 OpenJ9 warm cache + startup tuning + simple app
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1643980475726] [levelValue: 800] Payara Micro  5 #badassmicrofish (build 879) ready in 8,042 (ms)

Source: https://blog.openj9.org/2019/10/22/running-payara-micro-on-openj9/amp/?__twitter_impression=true
JVM - Behavioural Differences
Switching to a different JVM… you can do that… easily?

https://adoptopenjdk.net

Download for macOS x64

1. Choose a Version
   - OpenJDK 8 (LTS)
   - OpenJDK 11 (LTS)
   - OpenJDK 16 (Latest)

2. Choose a JVM
   - HotSpot
   - OpenJ9

AdoptOpenJDK has moved…
But why go through all this trouble?

**Memory Benefits**
- Save memory which can significantly reduce your bill for running in a cloud

**Startup Benefits**
- Faster development by shaving off seconds while starting application servers
- Faster startup of applications while scaling up in a cloud
Application Server Support

A designated JVM can be provided, recommended or enforced by your Application Server vendor if you make use of a Support Contract.
Deploy
Docker

```bash
INFO: BUILD SUCCESS
INFO: Total time: 2.255 s
INFO: Finished at: 2020-07-10T13:02:39+02:00
INFO: 

+ hollow-jar-project git:(solution) docker build . -t hollow-jar-project
Sending build context to Docker daemon 62.98kB
Step 1/2: FROM payara/micro:5.193
  --> c390e956cd7f
Step 2/2: COPY ./target/hollow-jar-project.war $DEPLOY_DIR
  --> cda89753b8fb
Successfully built cda89753b8fb
Successfully tagged hollow-jar-project:latest
+ hollow-jar-project git:(solution) docker history hollow-jar-project

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>cda89753b8fb</td>
<td>5 seconds ago</td>
<td>/bin/sh -c #nop</td>
<td>COPY file:dae6431a0401a5b5... 5.65kB</td>
<td></td>
</tr>
<tr>
<td>c390e956cd7f</td>
<td>10 months ago</td>
<td>/bin/sh -c wget</td>
<td>--no-verbose -O $PAYARA_HOME/... 79.4MB</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV PAYARA_VERSION=5.193 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ARG PAYARA_VERSION=5.193 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>CMD [&quot;-&quot;&quot;deplomentDir&quot; &quot;/&quot; 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENTRYPONT [&quot;java&quot; &quot;:XX:+ 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>WORKDIR /opt/payara 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>USER payara 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c addgroup payara &amp; 0B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV PAYARA_HOME=/opt/payara 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>10 months ago</td>
<td>/bin/sh -c #nop</td>
<td>EXPOSE 6300 8080 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>14 months ago</td>
<td>/bin/sh -c ZUUL ARCH=zulu8.38.0.13-ca-jdk8.0 208MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>14 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV JAVA_HOME=/usr/java/jv... 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>14 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV LC_ALL=en_US.UTF-8 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>14 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV LANGUAGE=en_US:en 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>14 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ENV LANG=en_US.UTF-8 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>15 months ago</td>
<td>/bin/sh -c #nop</td>
<td>CMD [&quot;/bin/sh&quot;] 0B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>15 months ago</td>
<td>/bin/sh -c #nop</td>
<td>ADD file:2e3a37783f56a4a27... 5.53MB</td>
<td></td>
</tr>
</tbody>
</table>
```
Docker Image

Ship your dependencies as part of the infrastructure
- Download and ship your project's dependency together with the Docker Image

```
FROM payara/micro:5.2826.7-jdk11

# Downloads the Apache Derby Client library

# Copy an application to be loaded
COPY ./target/employee-event.war /opt/payara/deployments

```
Use docker-compose

- Ship anywhere
- Run anywhere
Health Probes
- Liveness = /health/live
- Readiness = /health/ready

- JAX-RS
- CDI
- JPA
- JSON-B

- JAX-RS
- MicroProfile RestClient
- MicroProfile FaultTolerance:
  - Retry, Fallback, Timeout

- JSF
- CDI
- MicroProfile RestClient

Database
Summary

Build
- Feature-rich programming model
- Build fast, ship less for optimal developer experience
- Packaging styles allow for various contextual advantages

Run
- Portability on application and runtime level

Deploy
- Fits in scalable environments with cloud-native capabilities
- Applicable for both monolithic and microservices architectures, and everything in between
References

Jakarta EE
- https://jakarta.ee
- https://jakarta.ee/compatibility
- https://github.com/eclipse-ee4j

MicroProfile
- https://microprofile.io
- https://github.com/eclipse/microprofile

Mailing Lists
- https://accounts.eclipse.org/mailing-list/jakarta.ee-community
- https://accounts.eclipse.org/mailing-list/microprofile-dev

2021 Jakarta EE Developer Survey Report
- https://accounts.eclipse.org/documents
Thank you!