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

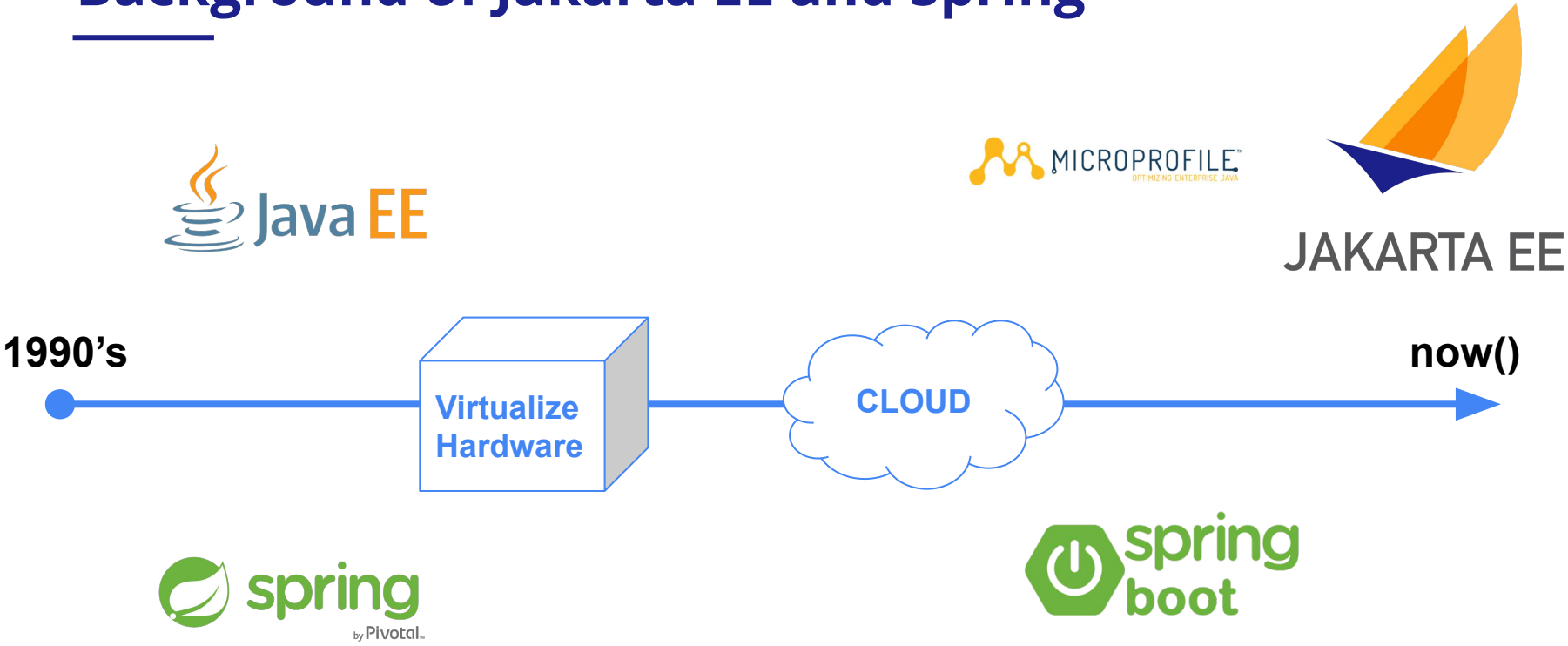
Covers the concepts, tools and tweaks that optimize building and running Jakarta EE / MicroProfile applications from a developer's perspective

Less code examples / Live coding

Background

JAKARTA EE

Background of Jakarta EE and Spring



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 can~~not~~ be used for building microservices! **@see MicroProfile**

Optimize Jakarta EE / MicroProfile enterprise apps

BUILD

- Programming Model
- Portability
- Build artifacts
- Developer Experience

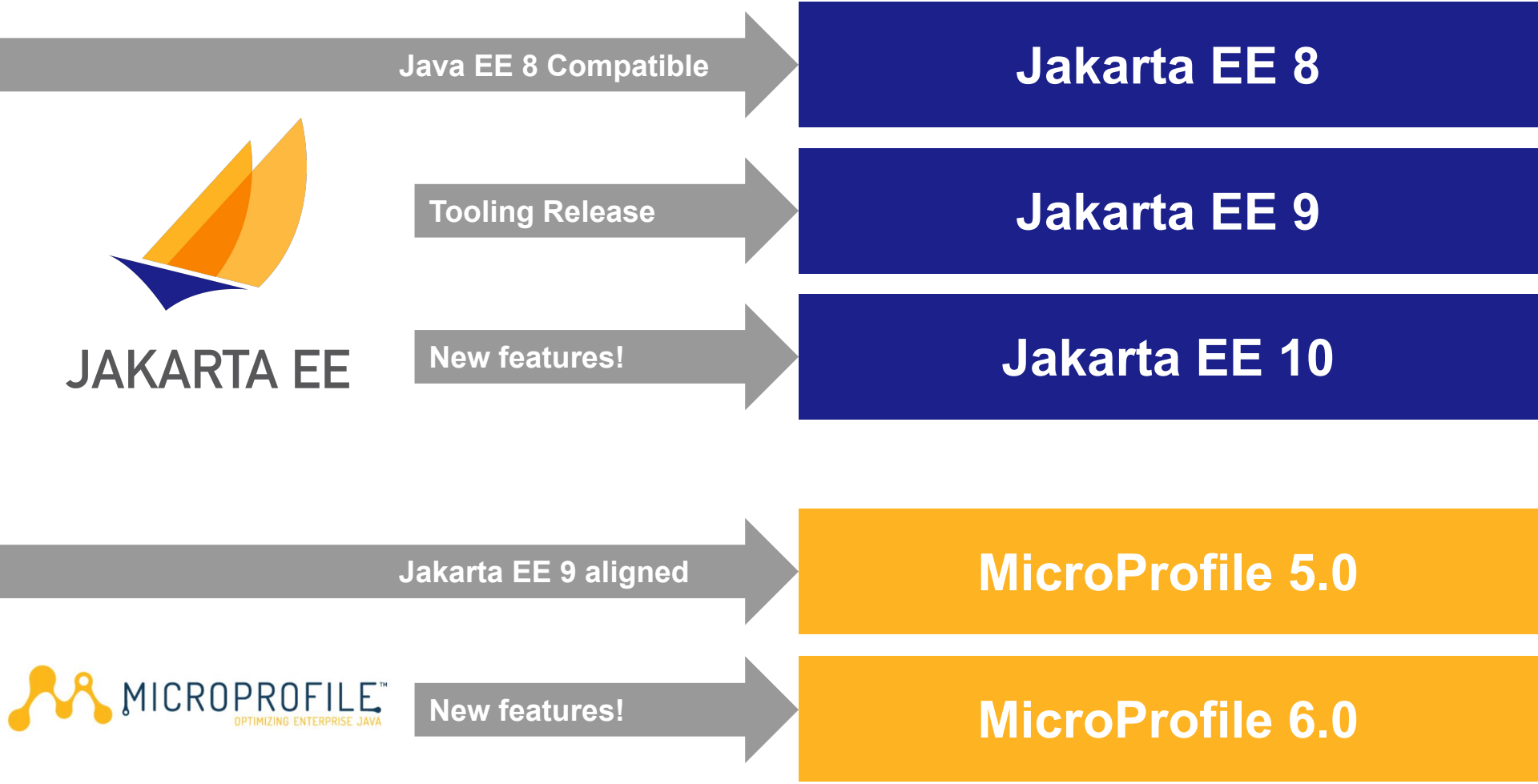
RUN

- Application Servers & Runtimes
- Influenced by Java SE

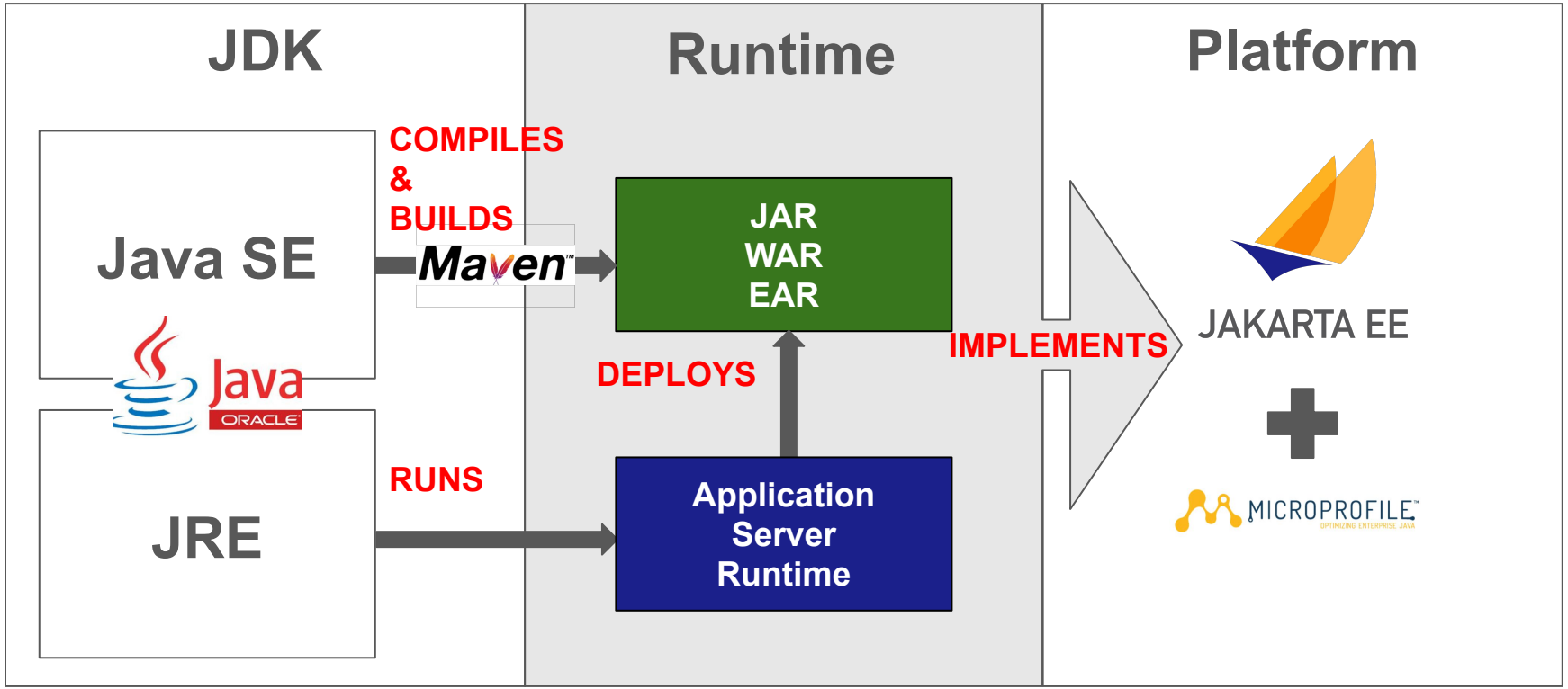
DEPLOY

- Separation of business logic and infrastructure
- Cloud Native Capabilities

Build



Overview



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>
        …..
```

Optimize building enterprise applications

Maven™

Thin WAR

A few Kilobytes
Only your business logic*

Fat WAR

Several Megabytes
Your business logic + dependencies

Uber / Fat JAR

Often lots of Megabytes
Your business logic + dependencies + runtime

Skimmed JAR

Several Kilobytes
Your business logic + runner

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.

1

Brian Vermeer
@BrianVerm

For a demo, I created a simple application with the Spring Boot scaffolder.
Unpacked all the jars, de-compiling all the classes, and counted the lines of code in my application.

Result:
Lines of code written: 4

Jars: 33
Class files: 9917
Lines of code: 518772

5:26 PM · Jun 25, 2020 · Twitter Web App

23 Retweets **92** Likes

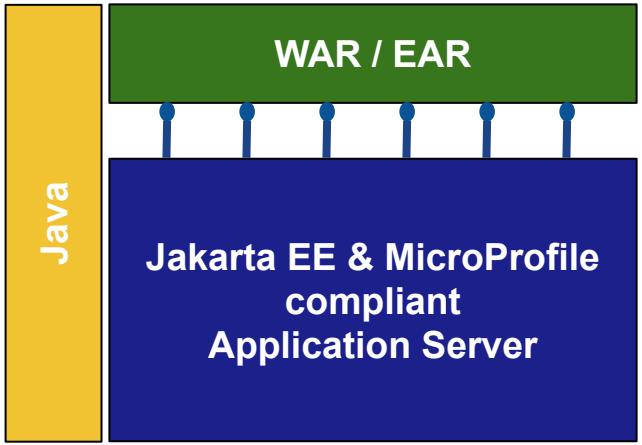
Run

JAKARTA EE

Photo by Miguel Á. Padriñán from Pexels

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>
```

payara

- 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 COMPATIBLE

Jakarta EE 9.1 Platform Compatible Products

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

Eclipse GlassFish
Eclipse Foundation
6.1 RC1
6

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

ManageFish Server
ManageCat
6.1.0

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

Thunisoft Application Server
Beijing Thunisoft Information Technology Co.,Ltd
3

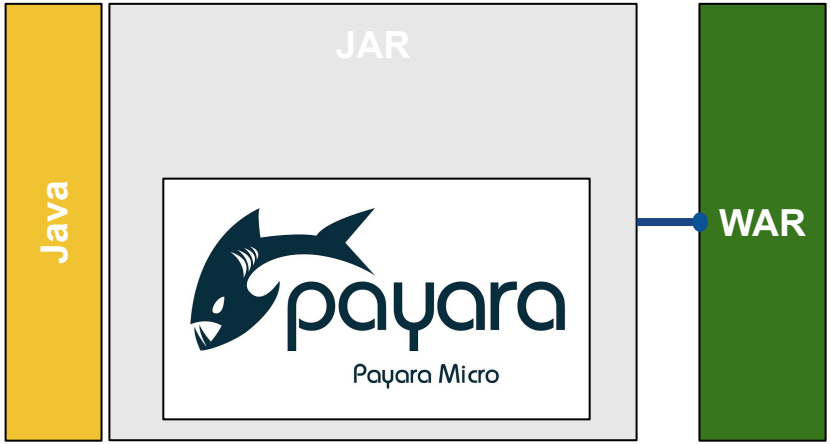
TongWeb Application Server
Beijing TongTech Co., Ltd.
9

WildFly
Red Hat
Preview 25.0.0.Final, Java 17
Preview 23.0.2.Final, Java 11
Preview 23.0.2.Final, Java 8

https://jakarta.ee/compatibility/

Hollow JAR

- Runs with Java version supported by the runtime



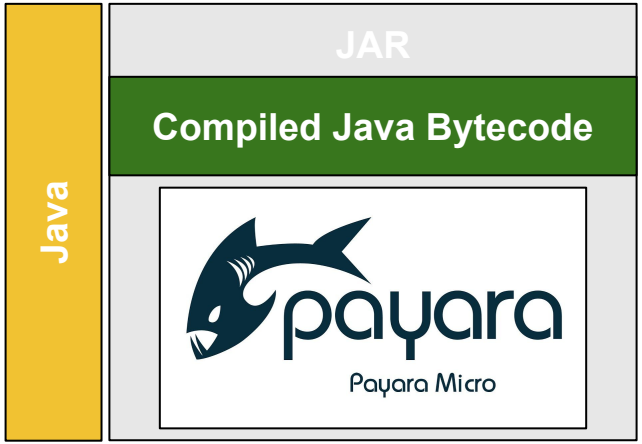
- Application server as one single artifact
- Little to no operational management tools
- Don’t include the whole application server with every build of your application
- “Just enough app server” for Thin and Fat WAR

MICROPROFILE™
OPTIMIZING ENTERPRISE JAVA

JAKARTA EE

Uber JAR

- Runs with Java version supported by the runtime



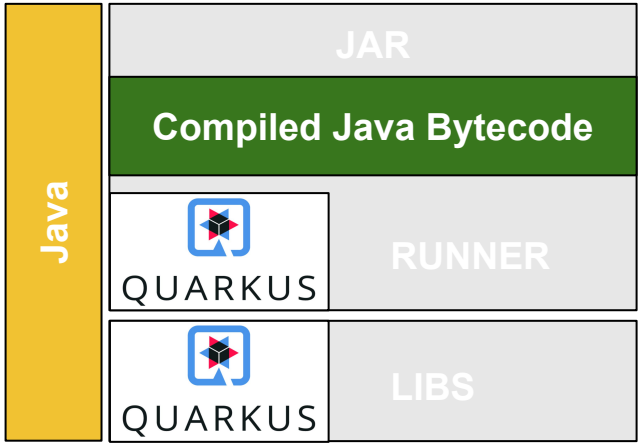
- One single artifact
- Little to no operational management tools
- Contains your code and application server
- Opposite of lean and tidy, so bigger in size
- “Just enough app server”

MICROPROFILE™
OPTIMIZING ENTERPRISE JAVA

JAKARTA EE

Skimmed JAR

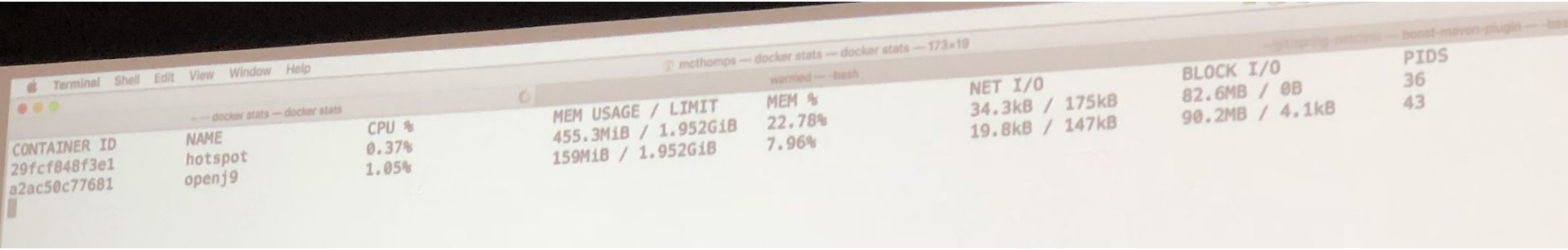
- Runs with Java version supported by the runtime



- 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)**

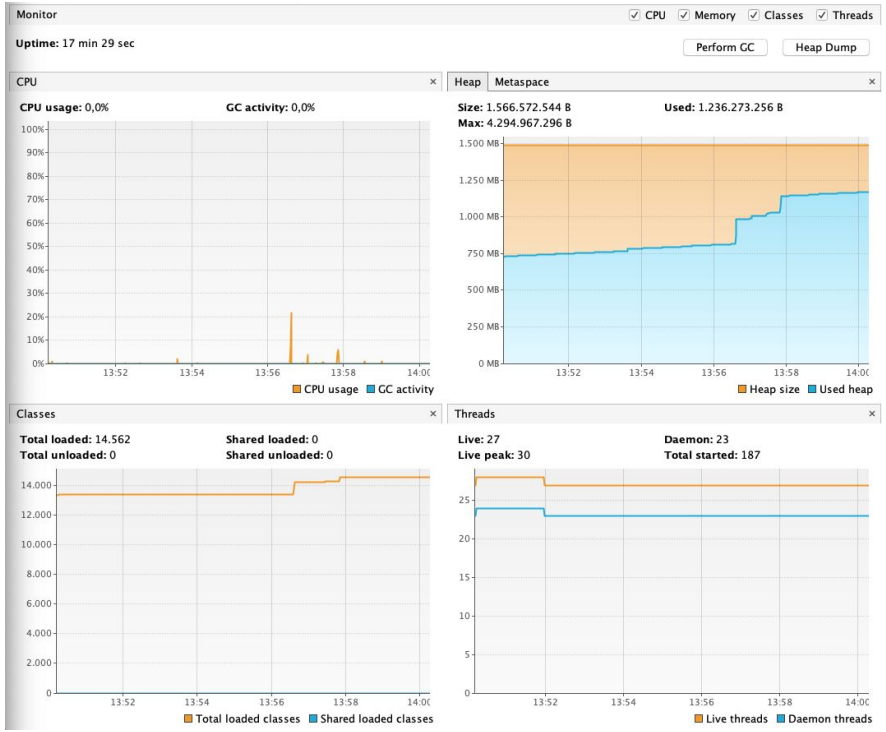
Payara Micro 5.2021.10 OpenJ9 warm cache + startup tuning
[INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1643979822119] [levelValue: 800] Payara Micro 5 #badassmicrofish (build 879) **ready in 3,853 (ms)**

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)**

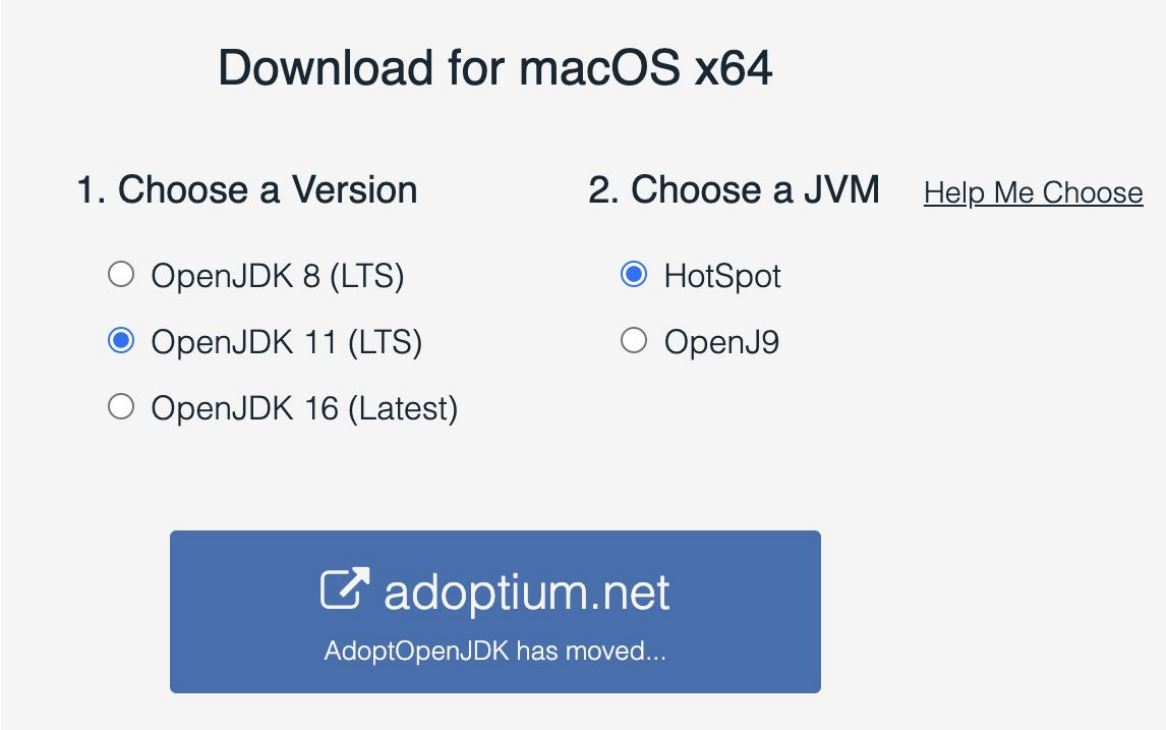
Source: https://blog.openj9.org/2019/10/22/running-payara-micro-on-openj9/amp/?__twitter_impression=true

JVM - Behavioural Differences



Switching to a different JVM… you can do that… easily?

https://adoptopenjdk.net



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

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[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
 ---> c350e956cd7f
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
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
cda89753b8fb        5 seconds ago       /bin/sh -c #(nop) COPY file:d0e6431a0401a5b5…   5.65kB
c350e956cd7f        10 months ago       /bin/sh -c wget --no-verbose -O ${PAYARA_HOM…   79.4MB
<missing>           10 months ago       /bin/sh -c #(nop)  ENV PAYARA_VERSION=5.193     0B
<missing>           10 months ago       /bin/sh -c #(nop)  ARG PAYARA_VERSION=5.193     0B
<missing>           10 months ago       /bin/sh -c #(nop)  CMD ["--deploymentDir" "/…   0B
<missing>           10 months ago       /bin/sh -c #(nop)  ENTRYPOINT ["java" "-XX:+…   0B
<missing>           10 months ago       /bin/sh -c #(nop) WORKDIR /opt/payara           0B
<missing>           10 months ago       /bin/sh -c #(nop)  USER payara                  0B
<missing>           10 months ago       /bin/sh -c addgroup payara &&     adduser -D…   5.03kB
<missing>           10 months ago       /bin/sh -c #(nop)  ENV PAYARA_HOME=/opt/paya…   0B
<missing>           10 months ago       /bin/sh -c #(nop)  EXPOSE 6900 8080             0B
<missing>           14 months ago       /bin/sh -c ZULU_ARCH=zulu8.38.0.13-ca-jdk8.0…   208MB
<missing>           14 months ago       /bin/sh -c #(nop)  ENV JAVA_HOME=/usr/lib/jv…   0B
<missing>           14 months ago       /bin/sh -c #(nop)  ENV LC_ALL=en_US.UTF-8       0B
<missing>           14 months ago       /bin/sh -c #(nop)  ENV LANGUAGE=en_US:en        0B
<missing>           14 months ago       /bin/sh -c #(nop)  ENV LANG=en_US.UTF-8         0B
<missing>           15 months ago       /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           15 months ago       /bin/sh -c #(nop) ADD file:2e3a37883f56a4a27…   5.53MB
➜  hollow-jar-project git:(solution)
```

Docker Image

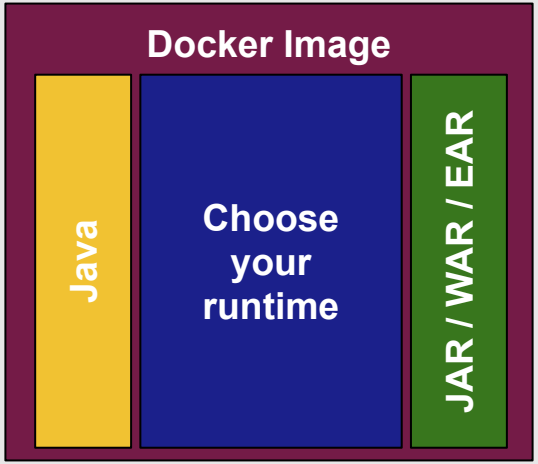
Ship your dependencies as part of the infrastructure

- Download and ship your project's dependency together with the Docker Image

```
FROM payara/micro:5.2020.7-jdk11

# Downloads the Apache Derby Client library
RUN wget -O /opt/payara/deployments/database-connector.jar https://search.maven.org/remotecontent?filepath=org/apache/derby/derbyclient/10.14.1.0/derbyclient-10.14.1.0.jar

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

ENTRYPOINT ["java", "-jar", "/opt/payara/payara-micro.jar", "--addJars", "/opt/payara/deployments/database-connector.jar", "--deploy", "/opt/payara/deployments/employee-event.war"]
```

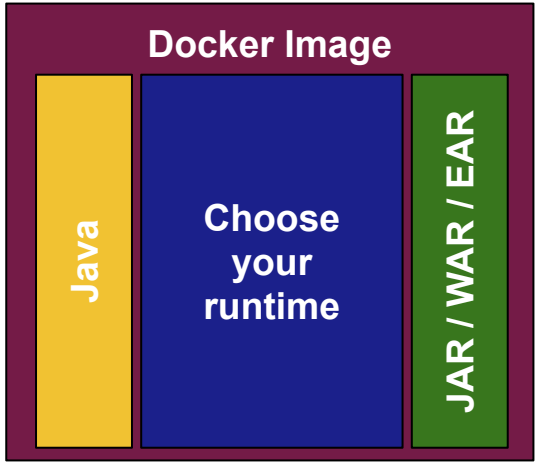
BUILD | RUN | Deploy

Operational Environment



kubernetes

docker



- Use docker-compose

- Ship anywhere
- Run anywhere

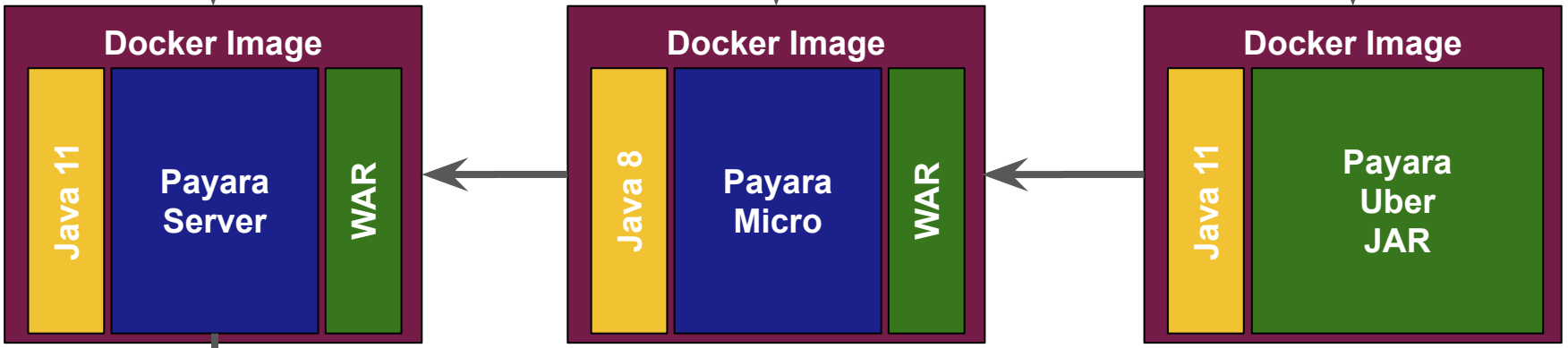
BUILD | RUN | Deploy

Health Probes

- Liveness = /health/live
- Readiness = /health/ready

kubernetes

- MicroProfile Health



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

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

- JSF
- CDI
- MicroProfile RestClient

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

JAKARTA EE

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

@edwinderks

cloudnativesolutions.guru

Thank you!

JAKARTA EE