

# Deployment options for OSGi applications in the cloud/edge

# Deployment options for OSGi applications in the cloud/edge

Speaker



**Dirk Fauth**

*Research Engineer  
Eclipse Committer*

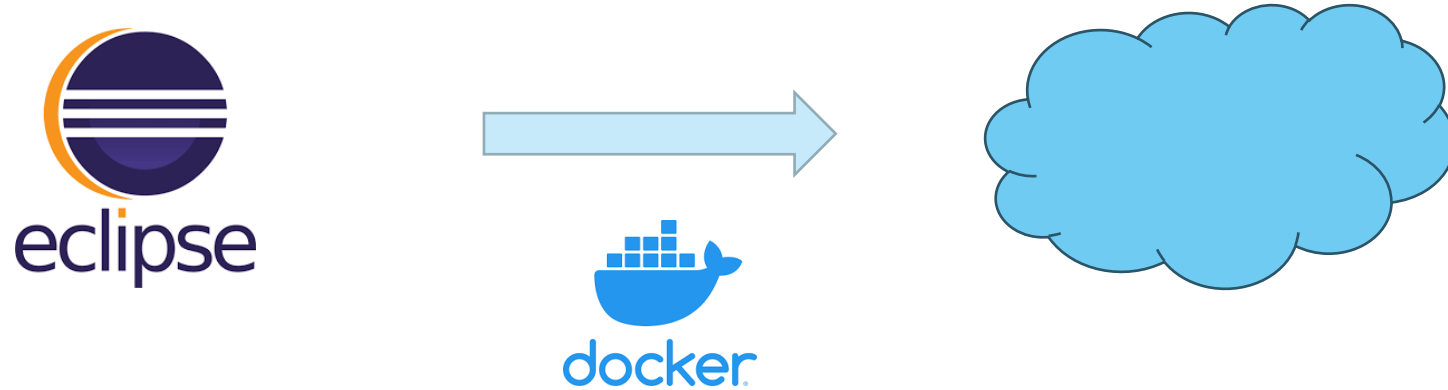
ETAS GmbH  
Borsigstraße 24  
70469 Stuttgart

[dirk.fauth@etas.com](mailto:dirk.fauth@etas.com)  
[www.etas.com](http://www.etas.com)

[blog.vogella.com/author/fipro/](https://blog.vogella.com/author/fipro/)  
Twitter: [fipro78](https://twitter.com/fipro78)

## Motivation

Shift existing Eclipse application (partly) to the cloud



*„Startup of a Java application to slow for cloud applications!“*

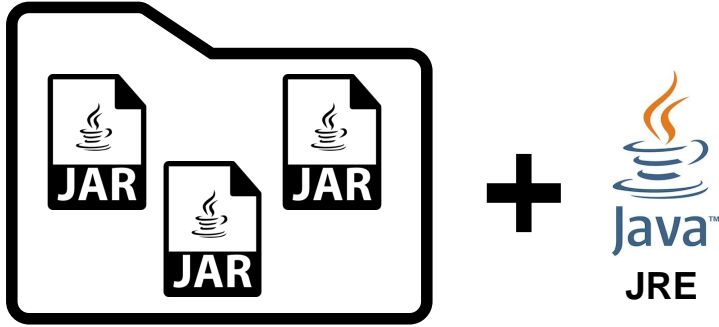
*„Size of a container for a Java application to big!“*

- 1. Deployment Variants**
- 2. Container**
- 3. Benchmark**
- 4. Conclusion**

# Deployment Variants

# Deployment Variants

## General



Multiple JARs in a folder



Executable JAR



Custom JRE (jlink)



Native Executable

# Deployment Variants

## Multiple JARs in a folder

- Multiple JAR files (OSGi bundles) inside a folder
- Additional configuration file
- Launcher

```
org.eclipse.osgi  
:org.eclipse.core.runtime.adaptor.EclipseStarter
```

```
java -jar org.eclipse.osgi-3.18.100.jar
```

- Build
  - maven-dependency-plugin
  - maven-resources-plugin

- ▼ app
  - ▼ configuration
    - config.ini
  - ▼ plugins
    - api-1.0.0-SNAPSHOT.jar
    - benchmark-1.0.0-SNAPSHOT.jar
    - command-1.0.0-SNAPSHOT.jar
    - configurable-1.0.0-SNAPSHOT.jar
    - eventhandler-1.0.0-SNAPSHOT.jar
    - impl-1.0.0-SNAPSHOT.jar
    - org.apache.felix.configadmin-1.9.26.jar
    - org.apache.felix.gogo.command-1.1.2.jar
    - org.apache.felix.gogo.runtime-1.1.6.jar
    - org.apache.felix.gogo.shell-1.1.4.jar
    - org.apache.felix.scr-2.2.6.jar
    - org.eclipse.equinox.event-1.6.200.jar
    - org.osgi.namespace.extender-1.0.1.jar
    - org.osgi.service.component-1.5.1.jar
    - org.osgi.service.event-1.4.1.jar
    - org.osgi.util.function-1.2.0.jar
    - org.osgi.util.promise-1.2.0.jar
    - org.eclipse.osgi-3.18.100.jar

<https://www.eclipse.org/equinox/documents/quickstart-framework.php>

# Deployment Variants

## Executable JAR

- Executable JAR that includes each required bundle as embedded JAR file
- Configuration also included in the executable JAR
- Launcher

```
aQute.launcher.pre.EmbeddedLauncher
```

```
java -jar equinox-app.jar
```

- Build
  - bnd-maven-plugin
  - bnd-export-maven-plugin



<https://bnd.bndtools.org/>

<https://bndtools.org/>

<https://github.com/bndtools/bnd/tree/master/maven-plugins>



# Deployment Variants

## Custom JRE via jlink

- Create a custom JRE with `jlink` command of the JDK
  - *assemble and optimize a set of **modules** and their dependencies into a custom runtime image*

<https://docs.oracle.com/en/java/javase/17/docs/specs/man/jlink.html>

- Folder layout like JRE
- Launcher: `java` command

```
java [options] -m <module>[/<mainclass>]
```

```
/app/jre $ ls -l
total 20
drwxr-xr-x 2 appuser appuser 4096 Oct 14 08:37 bin
drwxr-xr-x 4 appuser appuser 4096 Oct 14 08:37 conf
drwxr-xr-x 9 appuser appuser 4096 Oct 14 08:37 legal
drwxr-xr-x 4 appuser appuser 4096 Oct 14 08:37 lib
-rw-r--r-- 1 appuser appuser 140 Oct 14 08:37 release
/app/jre $
```

- Issue with OSGi and jlink  
Most available OSGi bundles do not contain a `module-info.class`  
→ **automatic module cannot be used with jlink**

# JPMS

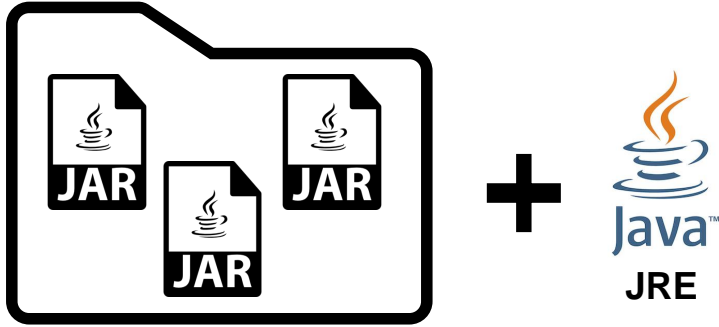
## Native Executable with GraalVM

- *Native Image is a technology to compile Java code ahead-of-time to a binary – a native executable. A native executable includes only the code required at run time, that is the application classes, standard-library classes, the language runtime, and statically-linked native code from the JDK.*
- Can be created using the GraalVM `native-image` tool
  - From a **Class**, a **JAR (classpath)** or a **Module (modulepath)**
- “Closed world assumption”
  - all the bytecode in your application that can be called at run time must be known at build time
- Issue with OSGi and `native-image`  
Dynamic classloading per bundle managed by OSGi Framework (Module Layer)  
**`java.lang.NullPointerException: A null service reference is not allowed.`**

<https://www.graalvm.org/reference-manual/native-image/>

# Deployment Variants

OSGi



Multiple JARs in a folder



Executable JAR



Custom JRE (jlink)



Native Executable

# Deployment Variants

## Custom JRE via jlink - OSGi

– Add `module-info.class`

– ModiTect

<https://github.com/moditect/moditect>

→ Intrusive change that adds an artifact to an existing published JAR

OSS license compatibility?

Checksum?

→ Requires knowledge on internals for generation

Maintenance?

– **Bndtools JPMS Support**

<https://bnd.bndtools.org/chapters/330-jpms.html>



Enable creation of `module-info.class` for each bundle, e.g. via `bnd-maven-plugin`

```
<plugin>
  <groupId>biz.aQute.bnd</groupId>
  <artifactId>bnd-maven-plugin</artifactId>
  <configuration>
    <bnd>
      <![CDATA[
Bundle-SymbolicName: ${project.groupId}.${project.artifactId}
-sources: true
-contract: *
-jpms-module-info:org.fipro.service.command;modules='org.apache.felix.configadmin'
-jpms-module-info-options: org.osgi.service.cm;ignore="true"
]]>
    </bnd>
  </configuration>
</plugin>
```

Enable creation of `module-info.class` for **executable jar** via `.bndrun` file

```
-jpms-module-info: \  
    ${project.groupId}.equinox.${project.artifactId};\  
    version=${project.version};\  
    ee=JavaSE-${java.specification.version}  
-jpms-module-info-options: jdk.unsupported;static=false
```

**This makes the executable jar itself a module!**

# Deployment Variants

## Custom JRE via jlink with Bndtools JPMS support

### Build

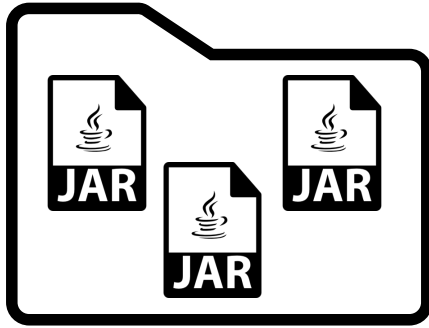
```
$JAVA_HOME/bin/jlink \  
  --add-modules org.fipro.service.equinox.app \  
  --module-path equinox-app.jar \  
  --no-header-files \  
  --no-man-pages \  
  --output /app/jre
```

### Launch

```
/app/jre/bin/java \  
  -m org.fipro.service.equinox.app/aQute.launcher.pre.EmbeddedLauncher
```

# Deployment Variants

OSGi



+



Multiple JARs in a folder



+



Executable JAR



Custom JRE (jlink)



Native Executable



## OSGi Connect

– *OSGi Connect allows for bundles to exist and be installed into the OSGi Framework from the flat class path, the module path (Java Platform Module System), a jlink image, or a native image.*

→ Allows to start an OSGi application without the full OSGi Module Layer

OSGi Core R8 – Connect Specification

<https://docs.osgi.org/specification/osgi.core/8.0.0/framework.connect.html>

Apache Felix Atomos

<https://github.com/apache/felix-atomos>

Ubiquitous OSGi - Android, Graal Substrate, Java Modules, Flat Class Path

<https://www.youtube.com/watch?v=KxmtzjHBumU>

OSGi R8, Felix 7, Atomos and the future of OSGi@Eclipse

<https://www.youtube.com/watch?v=oitFMbztf5s>

# Deployment Variants

## GraalVM Native Image with OSGi Connect

### – Preparation

1. Add/use Atomos to be able to start the OSGi application from the flat classpath
2. Generate reachability metadata via tracing agent (reflection, resources, ...)
3. Update generated metadata

### – Build

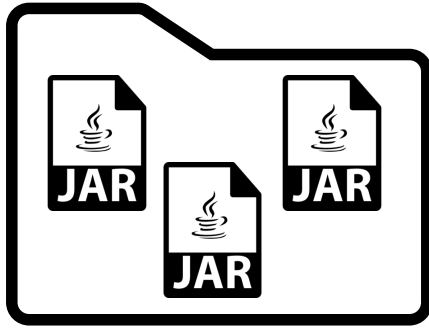
- Via GraalVM build plugins (Maven/Gradle)
- Docker multi-stage build using GraalVM container images

### – Notes/Remarks

- `native-image` build only worked with **flat classpath** and **listing all jars explicitly**
- Build result is platform-dependent
- `atomos_lib` folder or index file needed for Atomos to discover bundles and load bundle entries
- Still not everything is working as expected (e.g. `scr:list` produces an empty output)

# Deployment Variants

OSGi Connect / Apache Felix Atomos



+



Multiple JARs in a folder



+



Executable JAR



Custom JRE (jlink)

GraalVM™

Native Executable

## Deployment (plain OSGi)

Multiple JARs in folder

Executable JAR

Custom JRE (jlink)

GraalVM Native Image

## Deployment (OSGi Connect)

Multiple JARs in folder

Executable JAR

Custom JRE (jlink)

GraalVM Native Image

# Container

## “Size matters” – Find the right base image

### Alpine vs. Debian vs. Ubuntu

Image	Size
alpine:3	5.54 MB
debian:bullseye-slim	80.50 MB
ubuntu:jammy	77.84 MB

### Eclipse Temurin vs. IBM Semeru JDK vs. JRE

Image	Size
eclipse-temurin:17-jdk-jammy	~ 455 MB
eclipse-temurin:17-jdk-alpine	~ 356 MB
eclipse-temurin:17-jre-jammy	~ 266 MB
<b>eclipse-temurin:17-jre-alpine</b>	<b>~ 168 MB</b>
ibm-semeru-runtimes:open-17-jdk-jammy	~ 477 MB
ibm-semeru-runtimes:open-17-jre-jammy	~ 272 MB

## Interlude: Distroless

- *"Distroless" images contain only your application and its runtime dependencies. They do not contain package managers, shells or any other programs you would expect to find in a standard Linux distribution.*

Image		Size
gcr.io/distroless/static-debian11	minimal Linux for "mostly-statically compiled" languages that do not require libc	2.36 MB
gcr.io/distroless/base-debian11	minimal Linux, glibc-based system	20.32 MB
gcr.io/distroless/java17-debian11	base image plus OpenJDK 17 and its dependencies	230.88 MB

- Distroless Java image is based on Debian and glibc, therefore bigger than an Alpine Temurin image
- Can be interesting in production for security reasons, but not for size

<https://github.com/GoogleContainerTools/distroless>

## Java Best Practices

- Install only what you need
  - Use JRE instead of JDK
  - Use multi-stage builds (e.g. to create JRE or Native Image)
- Don't run Java apps as root
- Properly shutdown and handle events to terminate a Java application
- Take care of “container-awareness”

<https://snyk.io/blog/best-practices-to-build-java-containers-with-docker/>

<https://developers.redhat.com/articles/2022/04/19/java-17-whats-new-openjdk-container-awareness#>

<https://blog.openj9.org/2021/06/15/innovations-for-java-running-in-containers/>



## Building Docker Images

- Use dedicated Docker files instead of generation tools
- Integrate image creation as part of the build via `fabric8io/docker-maven-plugin`

### Maven/Gradle first

<https://github.com/fabric8io/docker-maven-plugin>

<http://dmp.fabric8.io/>

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <extensions>>true</extensions>
  <configuration>
    <images> ... </images>
  </configuration>
  <executions> ... </executions>
</plugin>
```

- Use multi-stage build to checkout sources and build in one container, then create new production container with build result only

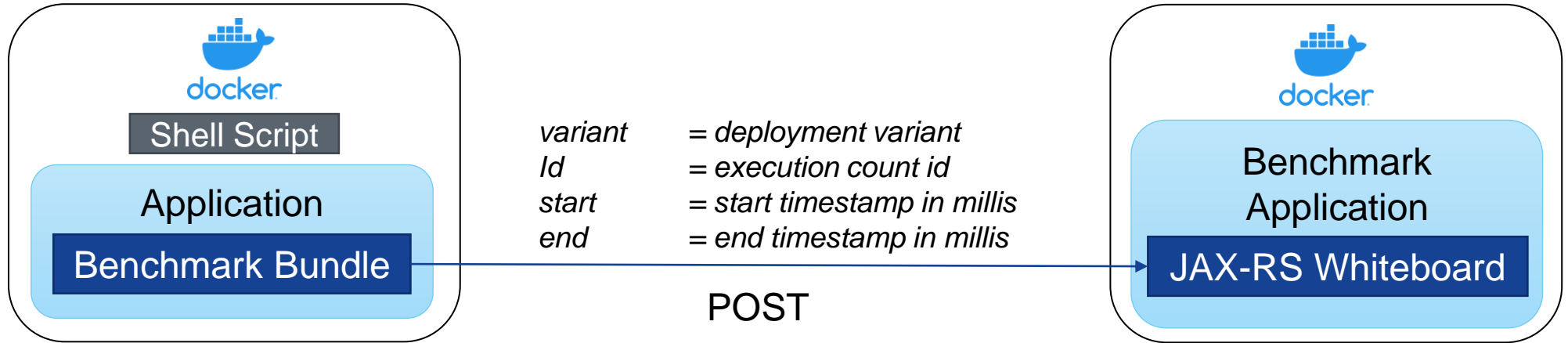
### Docker first

## Deployment Variant – Base Image – Image Size

Deployment (plain OSGi)	Base Image	Size
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 171 MB
Executable JAR	eclipse-temurin:17-jre-alpine	~ 174 MB
Custom JRE (jlink)	alpine:3	~ 75 MB
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB

Deployment (OSGi Connect)	Base Image	Size
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 171 MB
Custom JRE (jlink)	alpine:3	~ 75 MB
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB
GraalVM Native Image	scratch	~ 38 MB
	alpine:3	~ 43 MB

# Benchmark



## Benchmark Bundle / Immediate Component

- Get start timestamp from system property
- Get current timestamp
- Send POST request via `java.net.http.HttpClient`
- Shutdown

## Shell script

- Execute application multiple times in for-loop (clean/cache)
- Pass start timestamp as system property

# Benchmark Images

## Deployment Variant – Base Image – Image Size – Benchmark Image Size

Deployment (plain OSGi)	Base Image	Size	Size Benchmark
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 171 MB	~ 173 MB
Executable JAR	eclipse-temurin:17-jre-alpine	~ 174 MB	~ 176 MB
Custom JRE (jlink)	alpine:3	~ 75 MB	~ 78 MB
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB	~ 55 MB

- + coreutils
- + nanosecond support
- + benchmark bundle
- + java.net.http module
- + shell script support

Deployment (OSGi Connect)	Base Image	Size	Size Benchmark
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 171 MB	~ 173 MB
Custom JRE (jlink)	alpine:3	~ 75 MB	~ 78 MB
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB	~ 55 MB
GraalVM Native Image	scratch alpine:3	~ 38 MB (~ 43 MB)	(~ 46 MB) ~ 53 MB

# Benchmark Results

Deployment (plain OSGi)	Base Image	Size	Size Benchmark	Startup Clean	Startup Cache
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 171 MB	~ 173 MB	~ 982 ms	~ 901 ms
Executable JAR	eclipse-temurin:17-jre-alpine	~ 174 MB	~ 176 MB	~ 1087 ms	~ 1099 ms
Custom JRE (jlink)	alpine:3	~ 75 MB	~ 78 MB	~ 1336 ms	~ 1345 ms
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB	~ 55 MB	~ 1497 ms	~ 1505 ms

Deployment (OSGi Connect)	Base Image	Size	Size Benchmark	Startup Clean	Startup Cache
Multiple JARs in folder classpath modulepath	eclipse-temurin:17-jre-alpine	~ 171 MB	~ 173 MB	~ 1122 ms ~ 1194 ms	~ 973 ms ~ 1052 ms
Custom JRE (jlink)	alpine:3	~ 75 MB	~ 78 MB	~ 1439 ms	~ 1326 ms
Custom JRE (jlink/compressed)	alpine:3	~ 53 MB	~ 55 MB	~ 1593 ms	~ 1445 ms
GraalVM Native Image	scratch alpine:3	~ 38 MB (~ 43 MB)	(~ 46 MB) ~ 53 MB	- ~ 34 ms	- -

# Conclusion

- All Java deployment variants possible for OSGi applications via
  - Bndtools JPMS support
  - OSGi Connect (Felix Atomos)
- Different deployment variants have different startup & runtime behaviors  
Consider JIT vs. AOT compilation
- Make decision about variant dependent on the use case,  
e.g. short running executables in container vs. long running application servers
- Further optimizations possible by configuring the Java runtime,  
e.g. Container-awareness, Garbage Collection, *Checkpoint & Restore*, etc.



***“AOT like startup performance with JIT runtime performance and behaviour”***

## – **CRaC (Coordinated Restore at Checkpoint)**

- OpenJDK CRaC JDK
- Azul Zulu JDK with CRaC support
- Enable via `-XX:CRaCCheckpointTo=<path>`
- Create a checkpoint via `jdk.crac.Core` API
- Create a checkpoint via `jcmd <PID> JDK.checkpoint` from separate shell process

## – **Open Liberty InstantOn / OpenJ9 CRIU support**

- IBM Semeru (OpenJDK/OpenJ9)
- Enable via `-XX:+EnableCRIUSupport`
- Creating a checkpoint via `org.eclipse.openj9.criu.CRIUSupport` API

## Current issues

- **CRaC (Coordinated Restore at Checkpoint)**
  - Only Linux, currently no Alpine (musl) variant available
  - Building a custom JRE (jlink) currently not easily possible (need to manually copy criu to the system)
  - Containers with CRaC support not yet available out-of-the-box
  
- **Open Liberty InstantOn / OpenJ9 CRIU support**
  - Only Linux, no Alpine
  - Building a custom JRE (jlink)?
  - Creating a checkpoint only possible via API

– April 2023

**Azul Zulu builds with CRaC support** generally available for Java 17 on Linux

– June 2023

**Open Liberty InstantOn** out of beta with release 23.0.0.6

– June 2023

Oracle announced **Oracle GraalVM** distributed under **GraalVM Free License** with release 23.0

→ Oracle GraalVM Enterprise now available for free

Be careful: *“You do not charge Your licensees any fees”*

→ Container images not yet available

[https://github.com/fipro78/osgi\\_deployment\\_options](https://github.com/fipro78/osgi_deployment_options)

Thank you

Dirk Fauth

ETAS/ENA

[dirk.fauth@etas.com](mailto:dirk.fauth@etas.com)