Adopt Open J9 for Spring Boot performance!

Charlie Gracie
Michael Thompson

https://www.flickr.com/photos/teegardin/

IBM

Outline

- Part 1 – The economics of Cloud and Java
- Part 2 - Java for the Cloud… Open J9
- Part 3 – Demo
- Part 4 – Wrap up

Part 1 – The economics of Cloud and Java

In the Cloud footprint is king

OpenJ9

GB/hr

This is the new measurement for application cost

In the Cloud footprint is king

OpenJ9

- Myth: machines have plenty of RAM, so optimizing for footprint is not worthwhile

In the Cloud footprint is king

OpenJ9

- Reality: application footprint is very important to:
  - Cloud users: pay for resources
  - Cloud providers: higher app density means lower operational costs

In the Cloud footprint is king

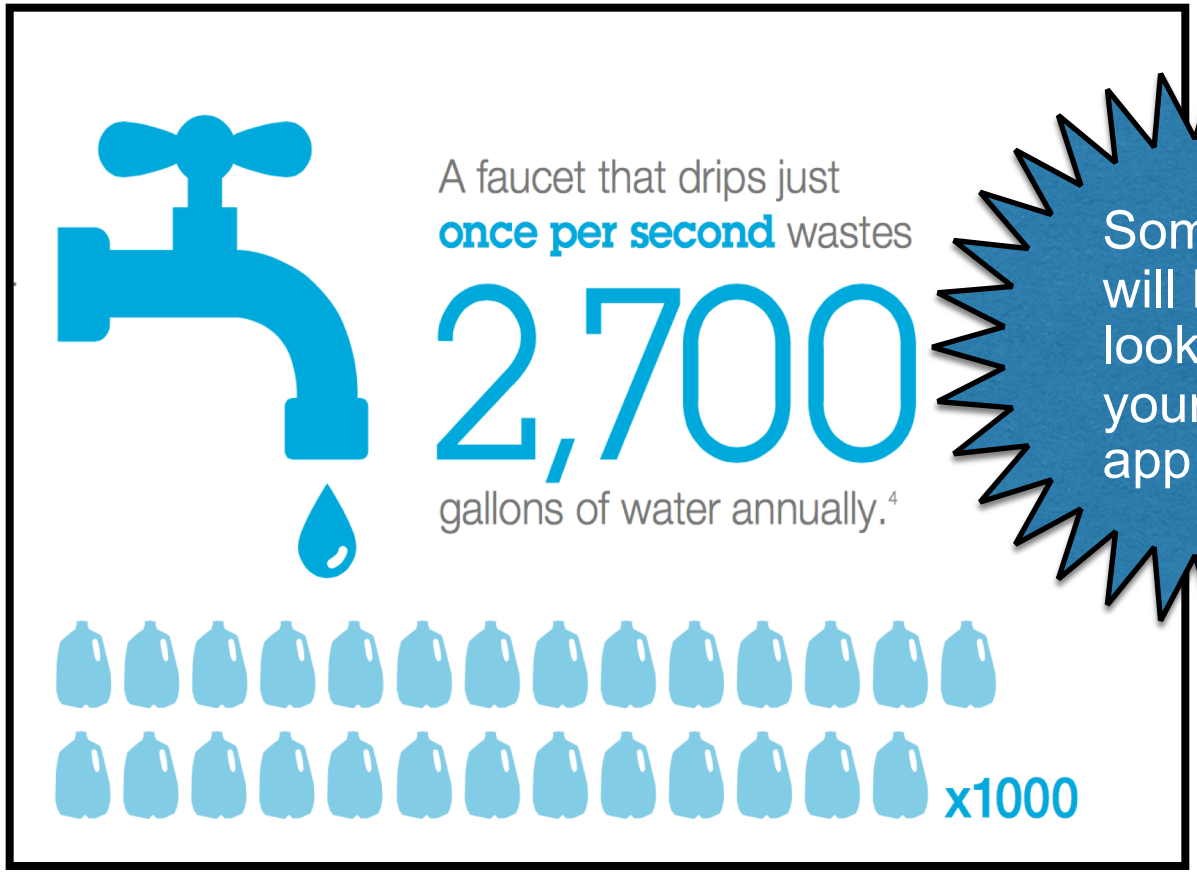
OpenJ9

- Reality: application footprint is very important to:
  - Cloud users: pay for resources
  - Cloud providers: higher app density means lower operational costs
- Trends:
  - Virtualization → big machines partitioned into many smaller VMs
  - Microservices → increased memory usage; native JVM footprint matters

In the Cloud footprint is king

OpenJ9

- Reality: application footprint is very important to:
  - Cloud users: pay for resources
  - Cloud providers: higher app density means lower operational costs
- Trends:
  - Virtualization → big machines partitioned into many smaller VMs
  - Microservices → increased memory usage; native JVM footprint matters
- Distinction between:
  - On disk image size – relevant for cloud providers , copy times
  - Virtual memory footprint – relevant for 32-bit applications
  - Physical memory footprint (RSS) relevant for real application costs



Someone will be looking at your leaky app

What does this mean to Cloud Java developers?

OpenJ9

- Changing –Xmx directly effects cost!
  - Very easy for businesses to understand

What does this mean to Cloud Java developers?

OpenJ9

- Changing –Xmx directly effects cost!
  - Very easy for businesses to understand
- Net effect: You'll be tuning your application to fit into specific RAM sizes
  - Smaller than you use today

What does this mean to Cloud Java developers?

OpenJ9

- Changing –Xmx directly effects cost!
  - Very easy for businesses to understand
- Net effect: You'll be tuning your application to fit into specific RAM sizes
  - Smaller than you use today
- You need to understand where memory is being used.
  - You'll be picking components based on memory footprint

What does this mean to Cloud Java developers?

OpenJ9

- Changing –Xmx directly effects cost!
  - Very easy for businesses to understand
- Net effect: You'll be tuning your application to fit into specific RAM sizes
  - Smaller than you use today
- You need to understand where memory is being used.
  - You'll be picking components based on memory footprint
- Increased memory usage for 1 service increases the bill by the number of concurrent instances!

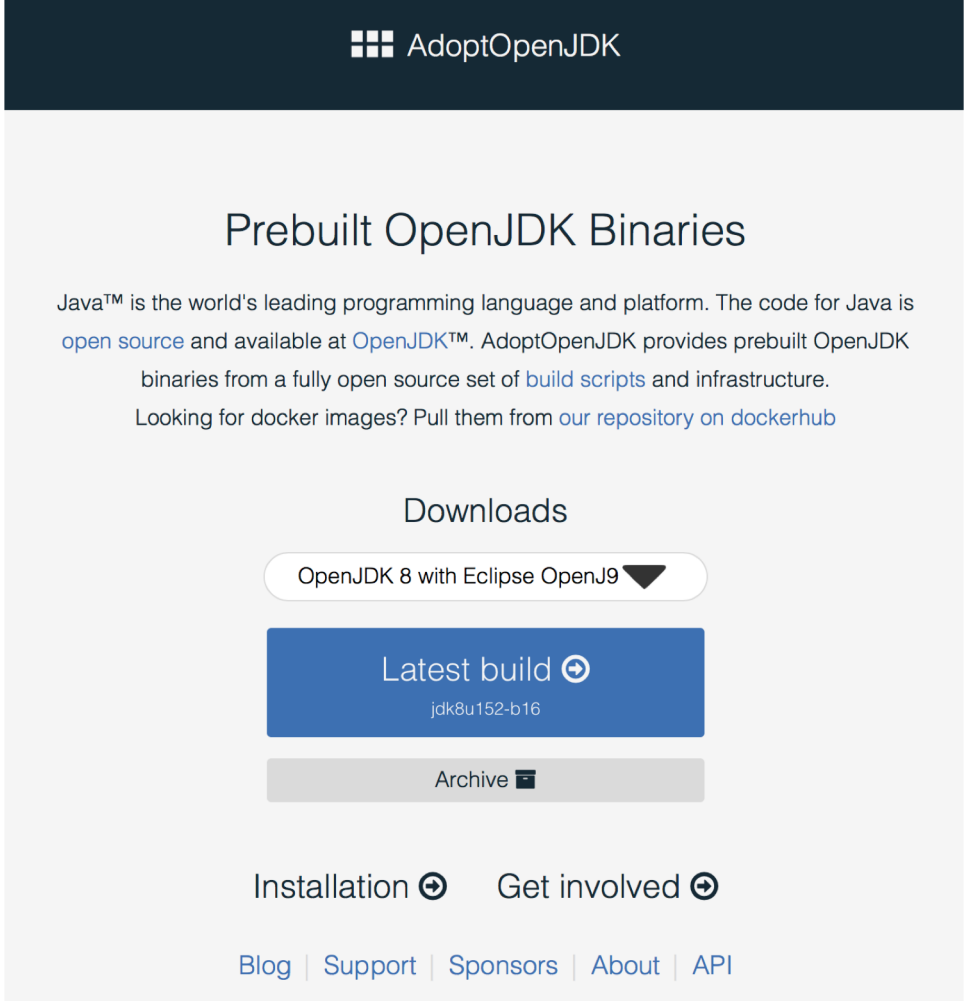
Part 2 - Java for the Cloud… Open J9

Eclipse OpenJ9
Created Sept 2017

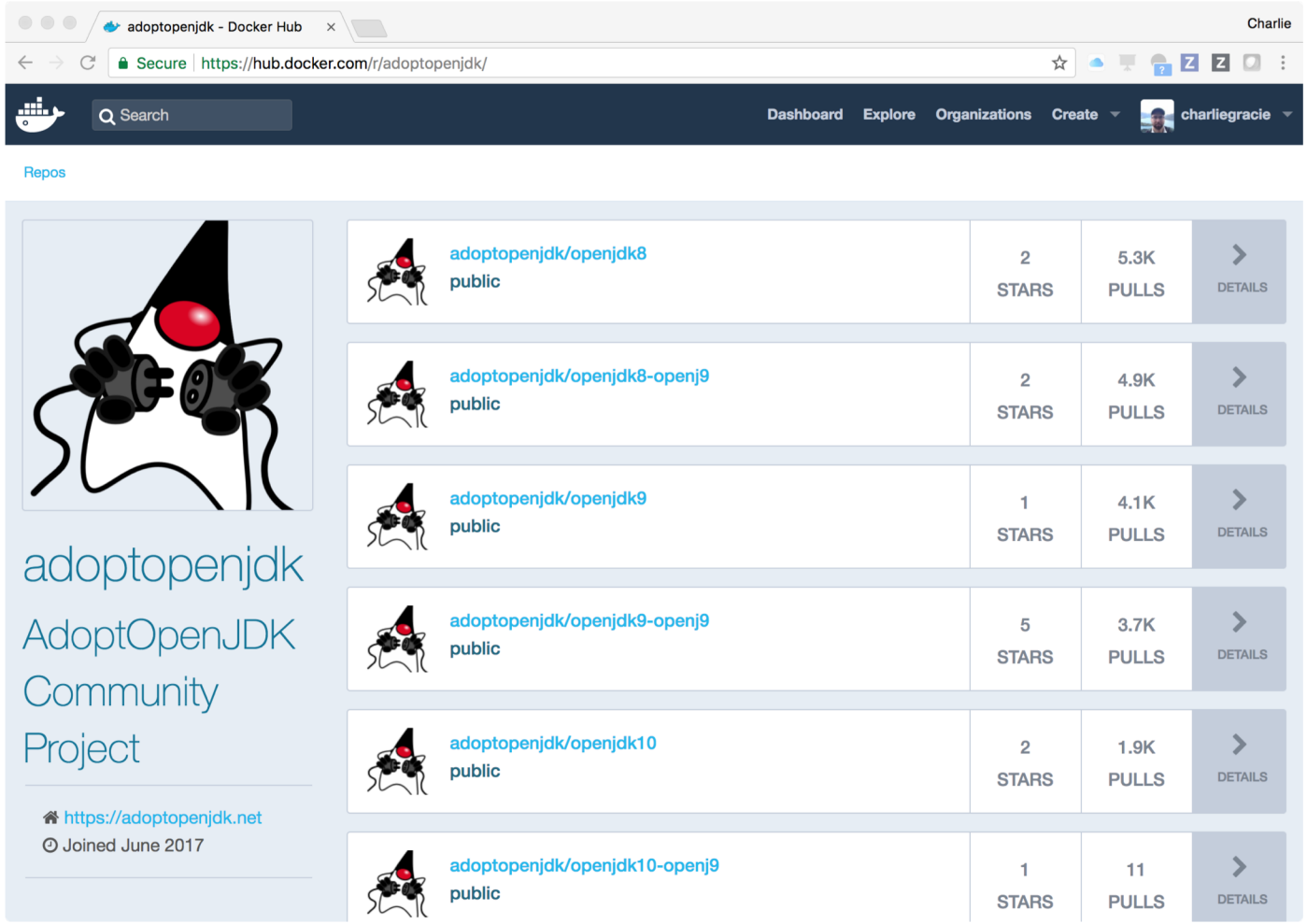
http://www.eclipse.org/openj9
https://github.com/eclipse/openj9

Dual License:
Eclipse Public License v2.0
Apache 2.0

Users and contributors very welcome
https://github.com/eclipse/openj9/blob/master/CONTRIBUTING.md



https://adoptopenjdk.net/?variant=openjdk8-openj9



https://hub.docker.com/r/adoptopenjdk/

Java ME Inside!

Java ME requirements

OpenJ9

- Small footprint
  - On disk and runtime.
  - Very limited RAM, usually more ROM
- Fast startup
  - Everybody wants their games to start quickly
- Quick / immediate rampup
  - Your game should not play better the longer you play

Java in the Cloud requirements

OpenJ9

- Small footprint
  - Improves density for providers
  - Improves cost for applications
- Fast startup
  - Faster scaling for increased demand
- Quick / immediate rampup
  - GB/hr is key, if you run for less time you pay less money

Java Heap and Garbage Collection

OpenJ9

- Smaller object sizes
  - Less overhead than other JVMs
- Innovative GC algorithms
  - Compact data structures use less memory
  - Aggressively use less heap

SharedClasses cache

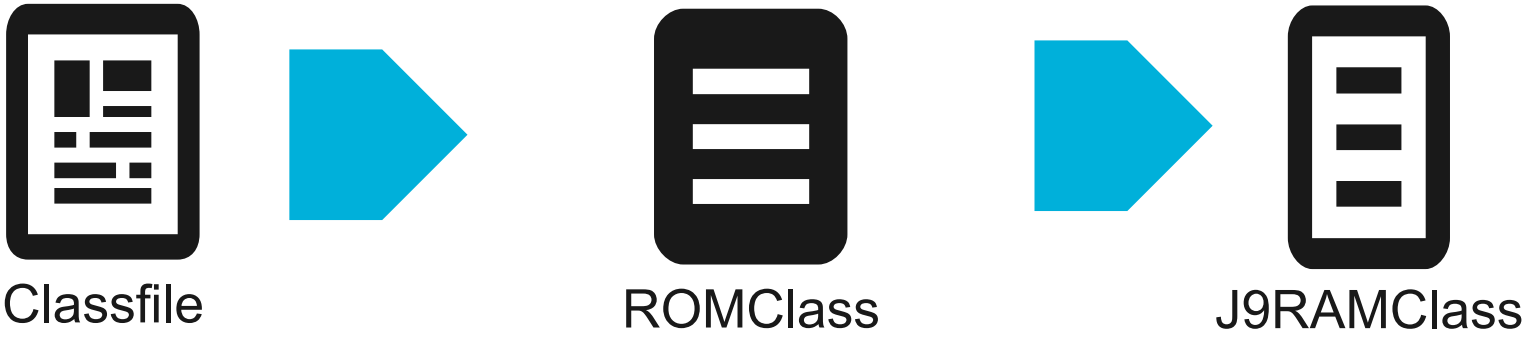
-Xshareclasses

    -enables the share classes cache

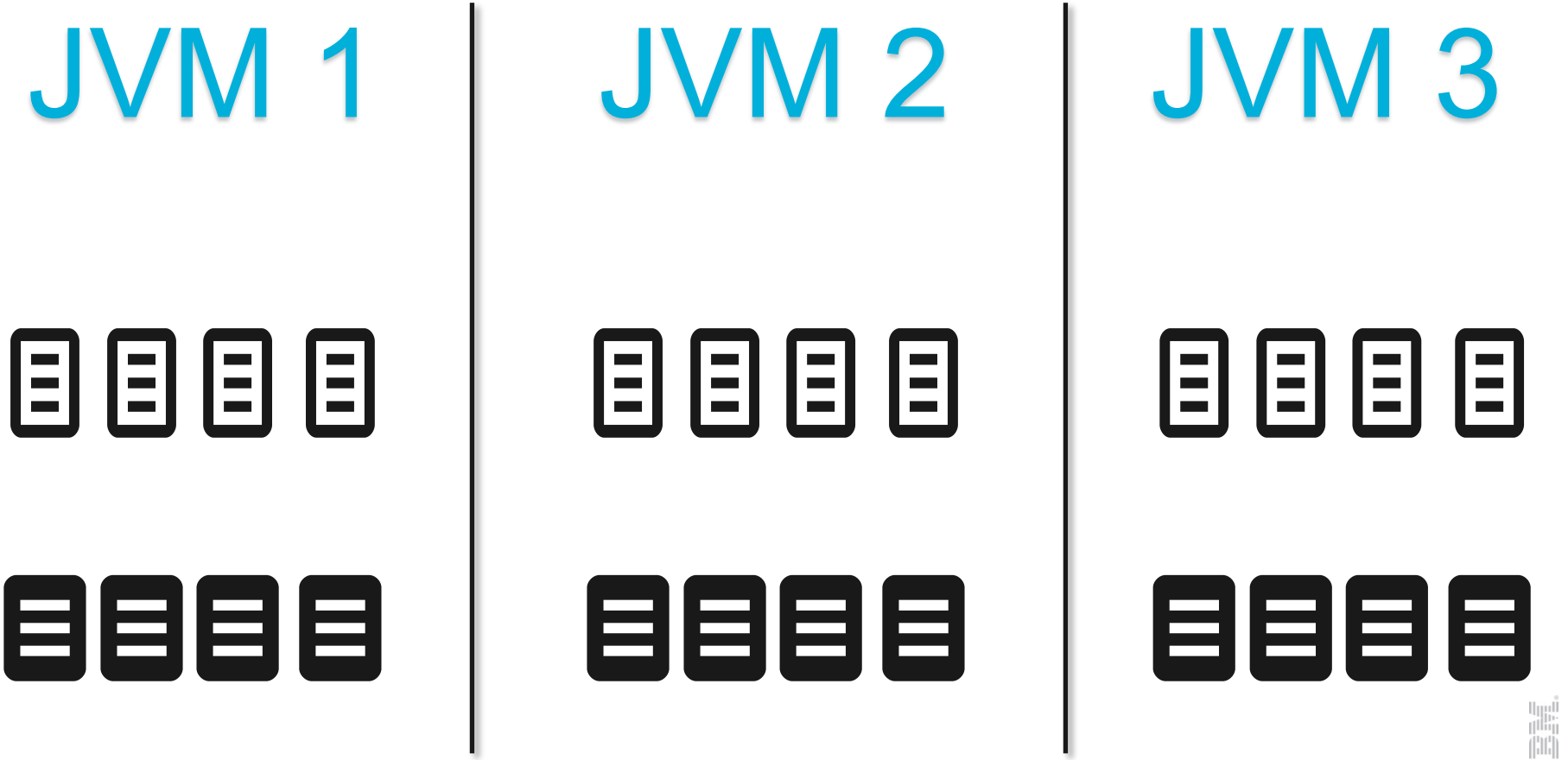
-Xscmx50M

    - sets size of the cache

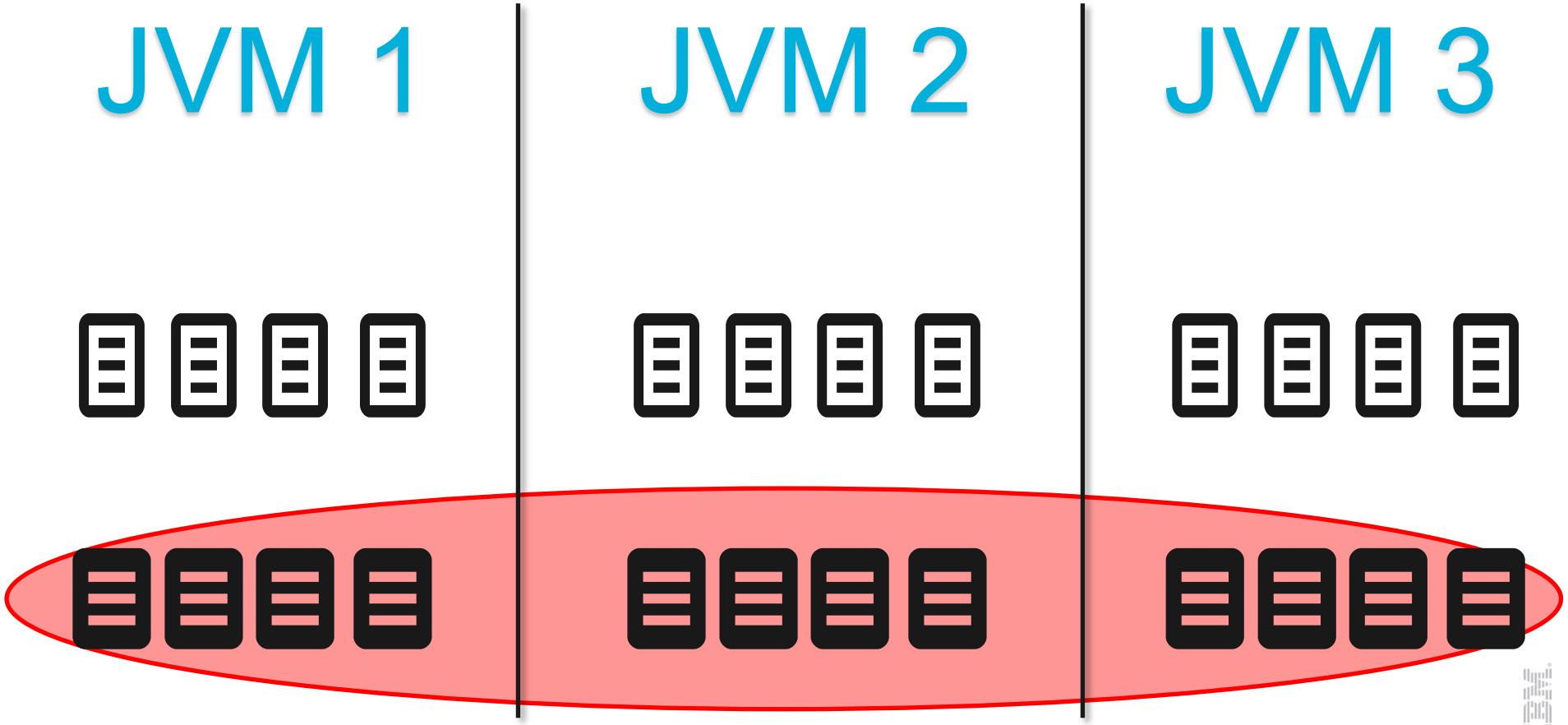
ShareClasses cache



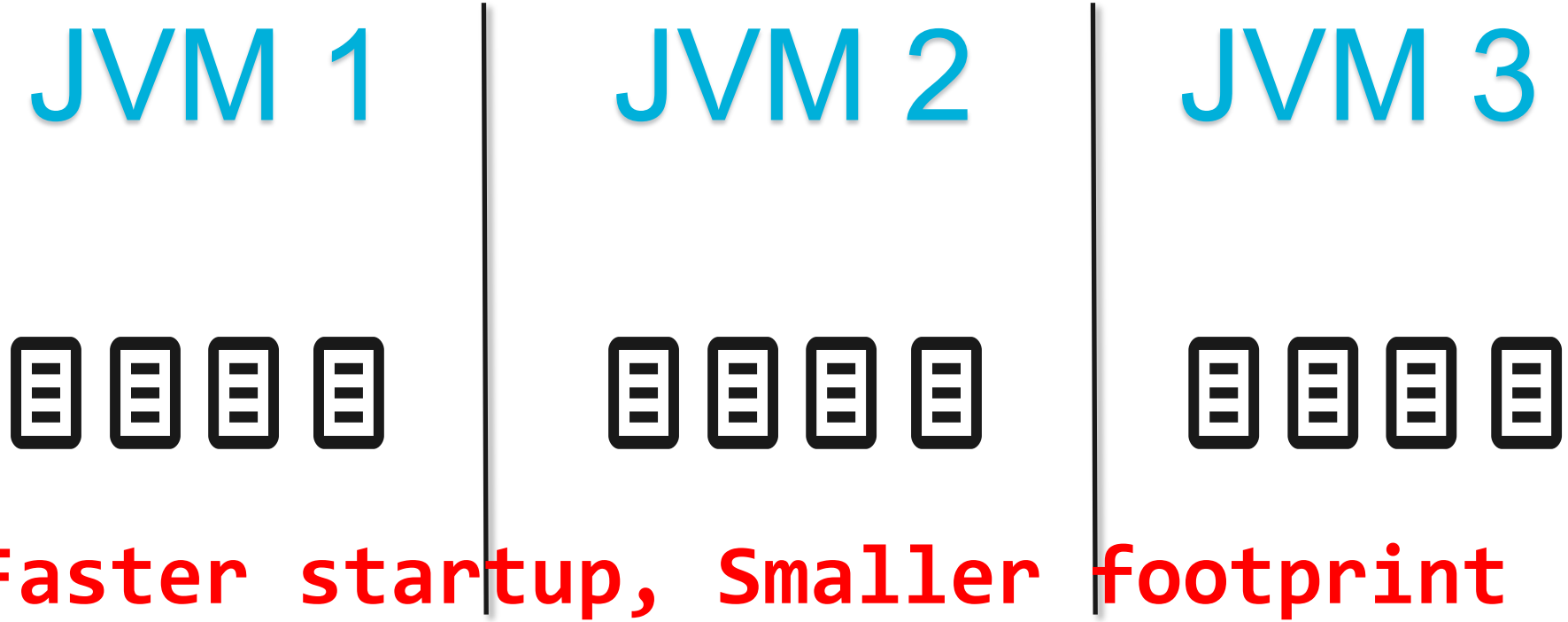
ShareClasses: ROM pays off



ShareClasses: ROM pays off



ShareClasses: ROM pays off



“Dynamic” AOT through ShareClasses


```
$ java -Xshareclasses ...
```

ShareClasses and AOT

OpenJ9

- Distinction between ‘cold’ and ‘warm’ runs
- Dynamic AOT compilation
  - –Relocatable format
  - –AOT loads are ~100 times faster than JIT compilations
  - –More generic code → slightly less optimized
    - Generate AOT code only during start-up
    - Recompilation helps bridge the gap

Further tuning options

- -Xquickstart
  - Designed for the fastest start-up
  - Ideal for short-lived tasks
  - May limit peak throughput
- -Xtune:virtualized
  - Tuning for containers
  - Enables VM idle management
  - Improves start-up and ramp-up. Trade-off of small throughput loss

Part 3 - Demo

Spring Boot w/ Eclipse OpenJ9

OpenJ9 – Benefits & Considerations

OpenJ9

Benefits:

- Simple to adopt (download & use)
- Smaller memory footprint
- Higher throughput
- Faster startup

OpenJ9 – Benefits & Considerations

OpenJ9

Benefits:

- Simple to adopt (download & use)
- Smaller memory footprint
- Higher throughput
- Faster startup

Considerations:

- Different –X arguments for tuning
- Different default GC algorithm

OpenJ9 – Benefits & Considerations

OpenJ9

Benefits:

- Simple to adopt (download & use)
- Smaller memory footprint
- Higher throughput
- Faster startup

Considerations:

- Different –X arguments for tuning
- Different default GC algorithm

As always, do your own testing!

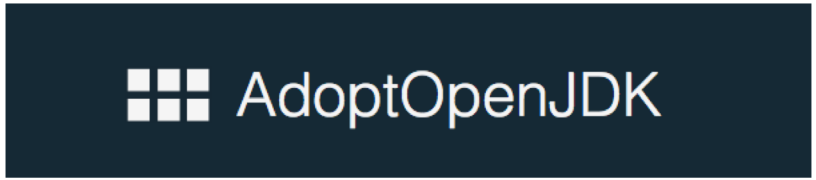
Get OpenJ9

Download from https://adoptopenjdk.net/

Docker base image:

Java 8 - https://hub.docker.com/r/adoptopenjdk/openjdk8-openj9/

Java 11 - https://hub.docker.com/r/adoptopenjdk/openjdk11-openj9/



Use OpenJ9

```
export JAVA_HOME=~/openjdk8-openj9/
export PATH=$PATH:$JAVA_HOME/bin
java –jar ...
```

Use OpenJ9 in Docker

OpenJ9

Docker File

```
FROM adoptopenjdk/openjdk8-openj9
...
CMD ["java","-jar",...]
```

🔥 Live Demo 🔥

Spring Boot w/ Eclipse OpenJ9

https://github.com/barecode/adopt-openj9-spring-boot

Spring Boot in Docker w/ OpenJ9

Docker File

```
FROM adoptopenjdk/openjdk8
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spring-projects/spring-petclinic.git
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-2.0.0.BUILD-SNAPSHOT.jar"]
```

Spring Boot in Docker w/ OpenJ9

OpenJDK w/ HotSpot

```
FROM adoptopenjdk/openjdk8
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spring-projects/spring-petclinic.git
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-2.0.0.BUILD-SNAPSHOT.jar"]
```

Spring Boot in Docker w/ OpenJ9

OpenJDK w/ OpenJ9

```
FROM adoptopenjdk/openjdk8-openj9
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spring-projects/spring-petclinic.git
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-2.0.0.BUILD-SNAPSHOT.jar"]
```

🔥 Live Demo 🔥

Spring Boot w/ Eclipse OpenJ9

https://github.com/barecode/adopt-openj9-spring-boot

Let’s go faster!

-Xquickstart
-Xshareclasses

JVM Options Refresher

OpenJ9

`-Xshareclasses`
- enables the share classes cache

`-Xscmx50M`
- sets size of the cache

`-Xquickstart`
- designed for the fastest start-up
- ideal for short-lived tasks
- may limit peak throughput

Spring Boot in Docker w/ OpenJ9

OpenJ9 with –Xquickstart & warmed –Xshareclasses

```
FROM adoptopenjdk/openjdk8-openj9
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spring-projects/spring-petclinic.git
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
RUN /bin/bash -c 'java -Xscmx50M -Xshareclasses –Xquickstart
    -jar spring-petclinic-2.1.0.BUILD-SNAPSHOT.jar &’ ; sleep 20 ;
    ps aux | grep java | grep petclinic | awk '{print $2}’ |
    xargs kill -1
CMD ["java","-Xscmx50M","-Xshareclasses","-Xquickstart",
     "-jar","spring-petclinic-2.1.0.BUILD-SNAPSHOT.jar"]
```

🔥 Live Demo 🔥

Spring Boot w/ Eclipse OpenJ9

https://github.com/barecode/adopt-openj9-spring-boot

Docker Layers Matter

(or why you should never do what Mike just did!)

How I created those images was stupid...

OpenJ9

Docker File

```
FROM adoptopenjdk/openjdk8
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spring-projects/spring-petclinic.git
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-2.0.0.BUILD-SNAPSHOT.jar"]
```


How I created those images was stupid...

OpenJ9

Docker File

```
FROM adoptopenjdk/openjdk8
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spr
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-
```

So many pointless layers!

Wasted size, image = 853MB

Fine for demos...

Terrible in the real world!

This is simpler...

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java",
    "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

This is better...

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG DEPENDENCY=target/dependency
COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app
ENTRYPOINT ["java","-cp","app:app/lib/*","hello.Application"]
```

But wait!

You said many layers were bad?

These layers are pointless

OpenJ9

Docker File

```
FROM adoptopenjdk/openjdk8
RUN apt-get update
RUN apt-get install -y \
    git \
    maven
WORKDIR /tmp
RUN git clone https://github.com/spr
WORKDIR /tmp/spring-petclinic
RUN mvn install
WORKDIR /tmp/spring-petclinic/target
CMD ["java","-jar","spring-petclinic-
```

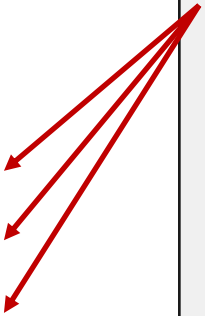
These layers don't help the app

Unused build artifacts and packages

The goal: create **lean** images

These layers are needed

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG DEPENDENCY=target/dependency
COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app
ENTRYPOINT ["java","-cp","app:app/lib/*","hello.Ap
```



The app pieces are the right layers

Split out for smaller layers & faster builds

The right layers matter ...

- Faster builds (cache re-use)

```
Step 1/10 : FROM adoptopenjdk/openjdk8-openj9
 ---> bf2da8bc5a91
Step 2/10 : RUN apt-get update
 ---> Using cache
 ---> 9582074cd6ef
```

- Faster deployents (less bits to push)

- Less wasted Docker repository space (reduced cloud costs)

How do I get there?

OpenJ9

Don’t include the build of the app in the final image!

Either build in the host OS

or...

Use multi-stage Docker build

Think about your layers

Approach may differ based on app
Different for Tomcat, Open Liberty, etc

Let **boost-maven-plugin** help you

Simplify the use of Docker for Spring Boot applications

pom.xml

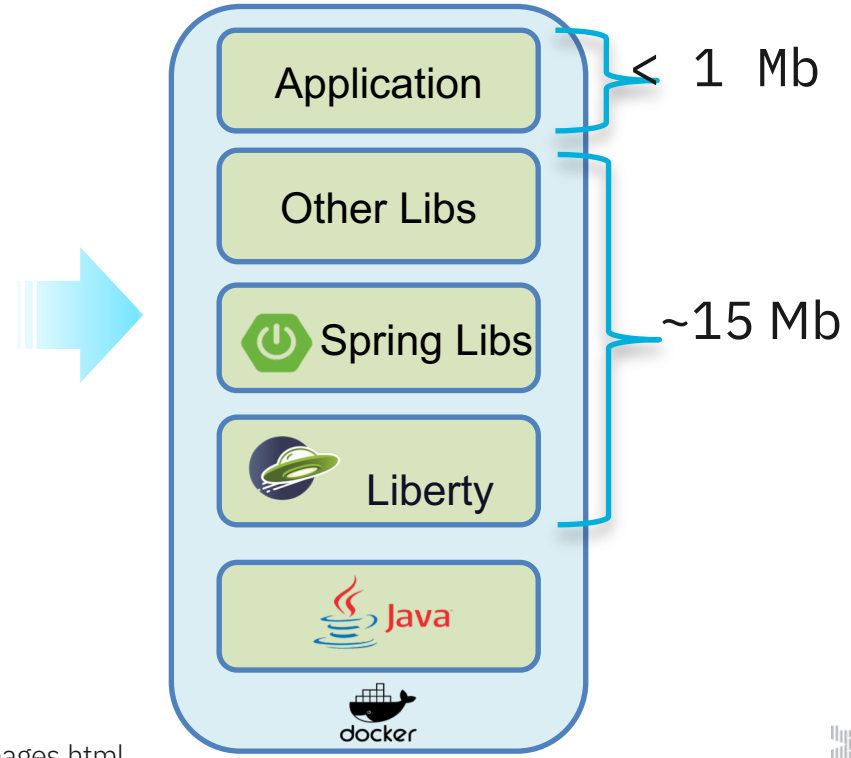
```
<plugin>
      <groupId>io.openliberty.boost</groupId>
      <artifactId>boost-maven-plugin</artifactId>
      <version>0.1</version>
</plugin>
```

Let **boost-maven-plugin** help you

OpenJ9

Boost creates the layers for you

```
mvn package boost:docker-build
```



https://openliberty.io/blog/2018/09/12/build-and-push-spring-boot-docker-images.html

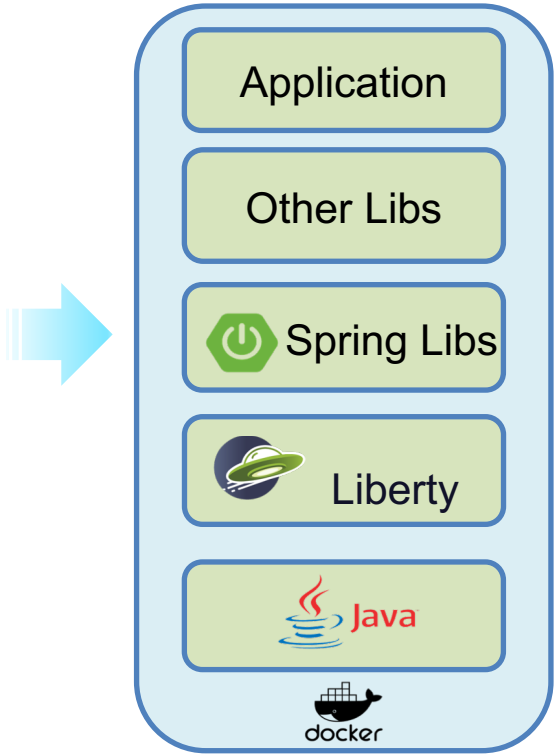
Let **boost-maven-plugin** help you

Boost creates the layers for you

pom.xml

```
<plugin>
 <!-- boost plugin -->
 <executions>
  <execution>
   <goals>
    <goal>docker-build</goal>
   </goals>
  </execution>
 </executions>
</plugin>
```

```
mvn package
```



OpenJ9

🔥 Live Demo 🔥

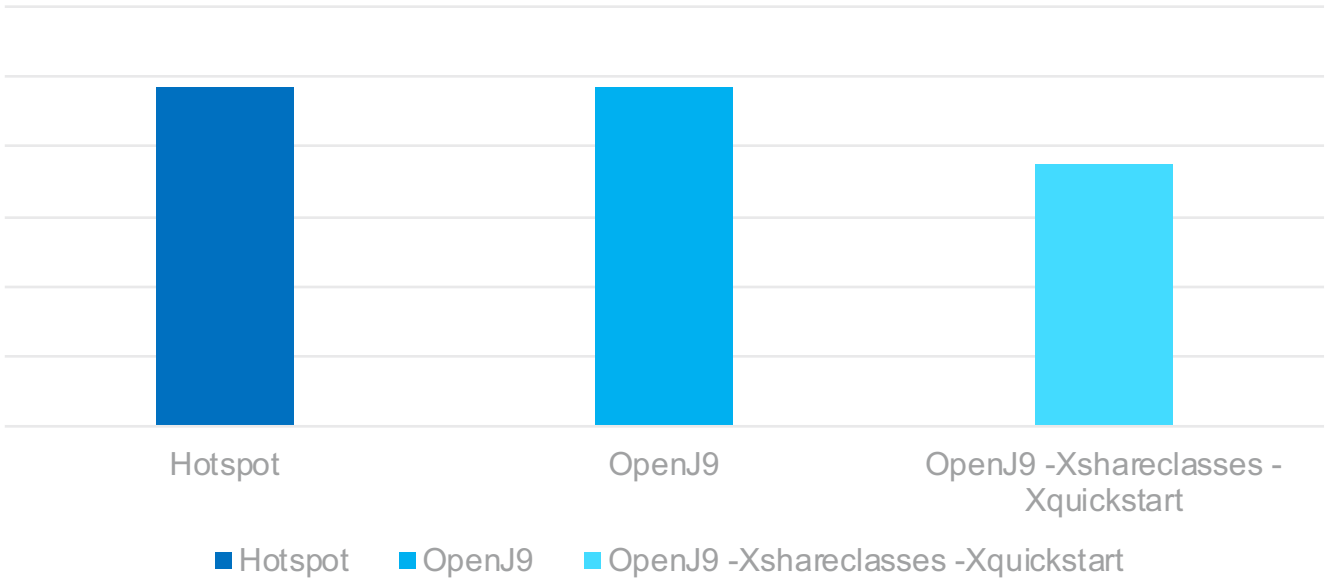
Spring Boot w/ Open Liberty & Eclipse OpenJ9

https://github.com/barecode/adopt-openj9-spring-boot

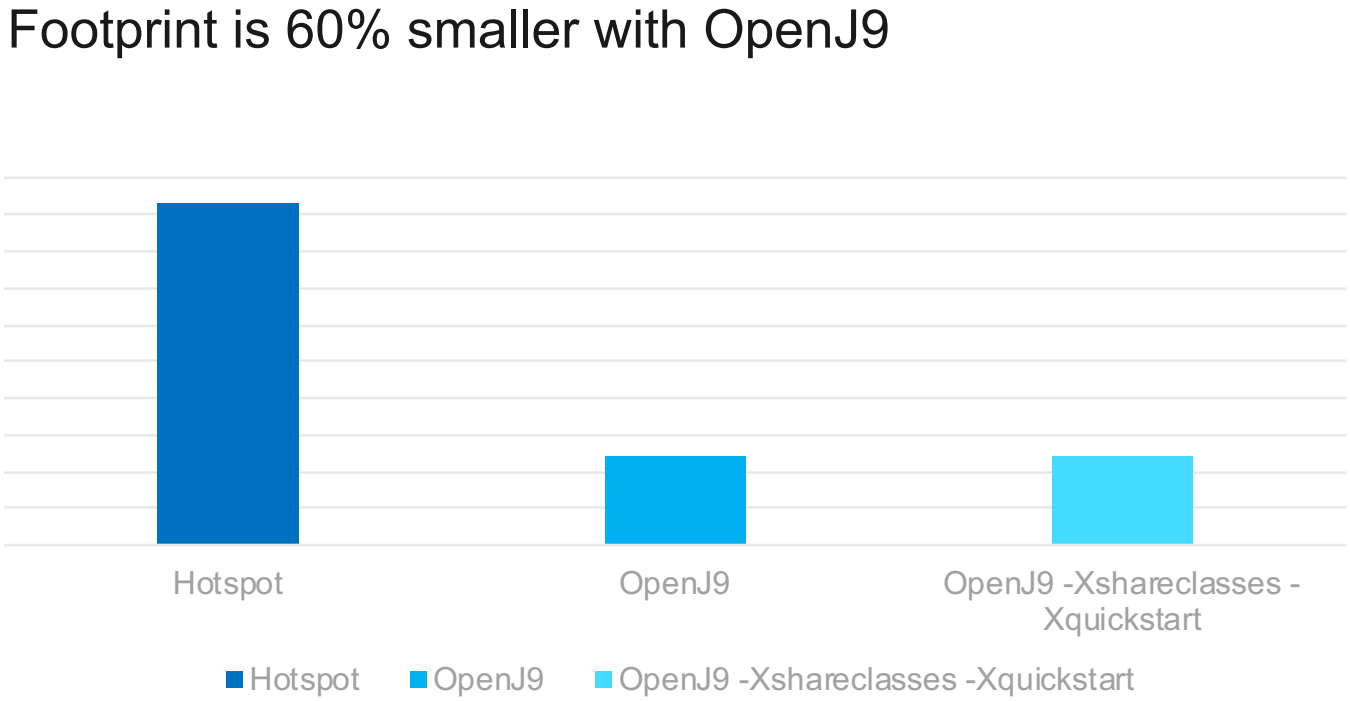
Part 4 – Wrap up

Results

Startup time is 30% faster with OpenJ9 –Xshareclasses -Xquickstart



Results


Footprint is 60% smaller with OpenJ9
Hotspot
OpenJ9
OpenJ9 -Xshareclasses -Xquickstart
Hotspot
OpenJ9
OpenJ9 -Xshareclasses -Xquickstart

Results

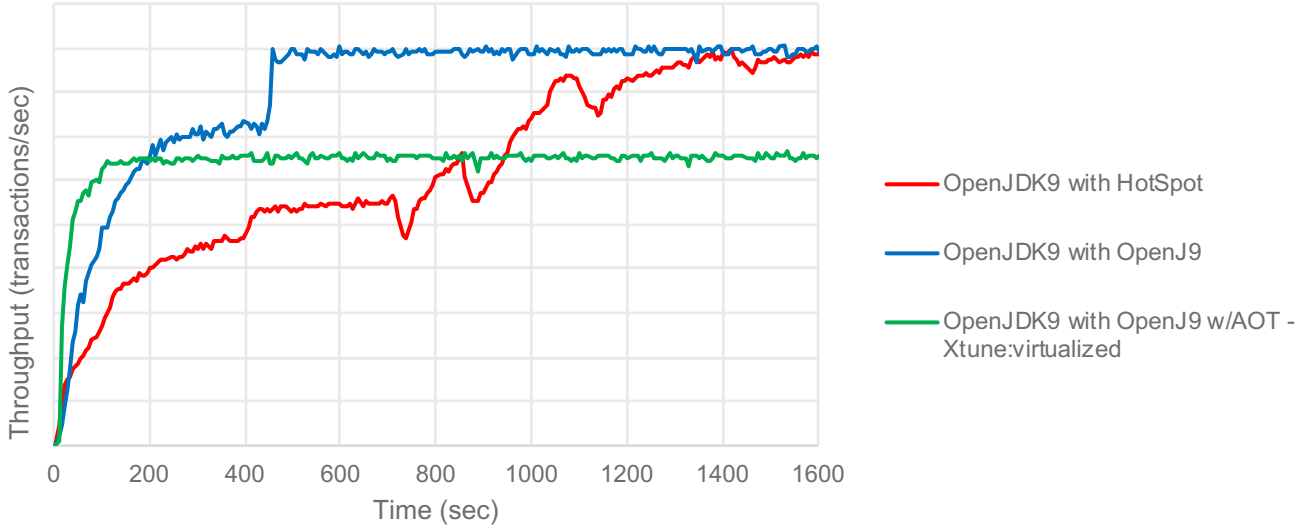
OpenJ9

OpenJ9 triggers ~55% fewer wakeups

- OpenJDK9 with HotSpot – 0.168% CPU
  - Summary: 84.7 wakeups/second, 0.0 GPU ops/seconds, 0.0 VFS ops/sec and 0.3% CPU use.
  - **Usage Events/s Category Description**
  - 0.9 ms/s 44.2 Process /sdks/OpenJDK9-x64_Linux_20172509/jdk-9+181/bin/java
  - 119.5 µs/s 20.0 Process [xfsaild/dm-1]
  - 138.6 µs/s 7.4 Timer tick_sched_timer
  - 10.5 µs/s 1.6 Process [rcu_sched]
  - 190.4 µs/s 1.5 Timer hrtimer_wakeup
- OpenJDK9 with OpenJ9 – 0.111% CPU
  - Summary: 38.5 wakeups/second, 0.1 GPU ops/seconds, 0.0 VFS ops/sec and 0.2% CPU use
  - **Usage Events/s Category Description**
  - 681.2 µs/s 19.2 Process /sdks/OpenJDK9-OPENJ9_x64_Linux_20172509/jdk-9+181/bin/java
  - 58.3 µs/s 5.2 Timer tick_sched_timer
  - 21.9 µs/s 3.6 Process [rcu_sched]
  - 39.3 µs/s 2.0 Timer hrtimer_wakeup
  - 157.1 µs/s 1.0 kWork ixgbe_service_task

Results

Ramping-up in a CPU constrained environment



-Xtune:virtualized and AOT good for CPU constrained situations and short running applications

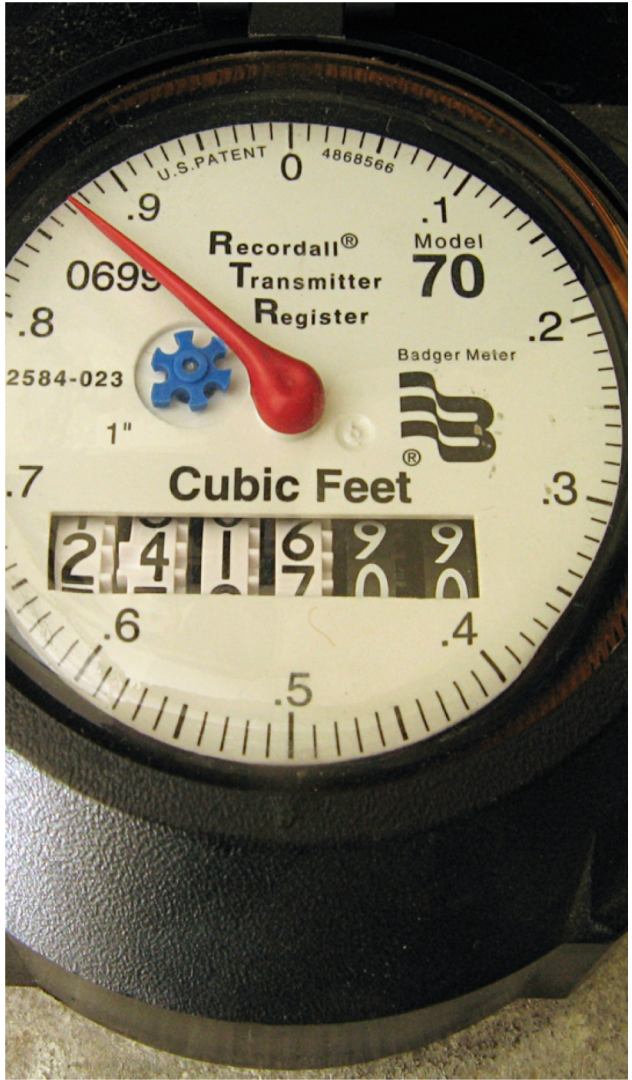
Its all change

How you design, code, deploy, debug, support etc will be effected by the metrics and limits imposed on you.

Financial metrics and limits always change behavior. It also creates opportunity

You will have to learn new techniques and tools

The JVM and Java applications have to get leaner and meaner



Thank you!