Distilling
Java Containers
We are Curios...

Have you packaged any app in a container?
We are Curios...

Have you packaged any app in a container?

What languages are you permitted to use?
We are Curios...

Have you packaged any app in a container?

What languages are you permitted to use?

What languages pique your interests?
"Nobody puts Java in a container"

Troubles with isolation of memory and CPU

Jörg Schad

- Head Of Engineering and Machine Learning at ArangoDB
- Mesosphere
Nobody puts Java in a container

Troubles with isolation of memory and CPU

Tl;dr: The Java Virtual Machine (not even with the Java 9 release) is not fully aware of the isolation mechanisms containers use internally, this can lead to unexpected behavior...


Jörg Schad

• Head Of Engineering and Machine Learning at ArangoDB
• Mesosphere
Nobody puts Java in a container

Troubles with isolation of **memory** and **CPU**

Tl;dr: The Java Virtual Machine (not even with the Java 9 release) is not fully aware of the isolation mechanisms containers use internally, this can lead to unexpected behavior...


---

Ken Sipe

- Cloud Systems Architect at D2IQ (Mesosphere)
- No Fluff Just Stuff Speaker

---

Jörg Schad

- Head Of Engineering and Machine Learning at ArangoDB
- Mesosphere
Distilling
Java Containers
Distillation Pattern
Distilling to get just the right taste, in a container
Distillation Pattern
Distilling to get just the right taste, in a container

PeatOps 😃
Distillation Pattern
Java Apps can be distilled farther than you might expect

Distilling Java App and its Container

While these upcoming techniques can be applied to other languages, Java remains a significant language for our business features. It’s important Java evolves with container ecosystems.
Distillation Pattern

8 Factors
Distillation Pattern

8 Factors

High cohesion
All features in the container are well defined, and used. “Do one thing and do it well.” -- Unix

Low coupling
Exposed interfaces are minimal, simple, and stable.
Distillation Pattern

8 Factors

High cohesion
All features in the container are well defined, and used. "Do one thing and do it well." -- Unix

Low coupling
Exposed interfaces are minimal, simple, and stable.

Immutable
Container not capable of or susceptible to change.

Idempotent
Multiple instances all behave the same, over time.
Distillation Pattern
8 Factors

High cohesion
All features in the container are well defined, and used. “Do one thing and do it well.” -- Unix

Low coupling
Exposed interfaces are minimal, simple, and stable.

Immutable
Container not capable of or susceptible to change.

Idempotent
Multiple instances all behave the same, over time.

Attack Phobic
Security attack vectors are resisted as potential surfaces have been minimized.

Small images
Less is more. Reduce I/O pressure. Images are revised, stored, transported, downloaded and scaled.
Distillation Pattern

8 Factors

**High cohesion**
All features in the container are well defined, and used. "Do one thing and do it well." -- Unix

**Immutable**
Container not capable of or susceptible to change.

**Attack Phobic**
Security attack vectors are resisted as potential surfaces have been minimized.

**Low coupling**
Exposed interfaces are minimal, simple, and stable.

**Idempotent**
Multiple instances all behave the same, over time.

**Rapid Startup**
Containers are embraced as ephemeral, fragile, expendable, and scalable cattle.

**Small images**
Less is more. Reduce I/O pressure. Images are revised, stored, transported, downloaded and scaled.

**Resource frugality**
Every bit and CPU tick is a limited resource as containers are scaled throughout your datacenter.
Distilling Java Containers
Java to Containers

Piqued Topics Related to Containers and Cloud Native
Java to Containers
Piqued Topics Related to Containers and Cloud Native

Cloud-native

- Approach to building and running applications that exploits advantages of cloud computing delivery model.

- How applications are created and deployed, not where.

- Pivotal
  https://pivotal.io/cloud-native
Milestones toward containers

Java 8

1/1996 → 3/2014
Milestones toward containers

Java 8

1/1996
3/2014
9/2017

Java 9

one feature release every six months
Milestones toward containers

Java 8
1/1996
3/2014

Java 9
9/2017

Java 10
3/2018

one feature release every six months
Milestones toward containers

1/1996

Java 8
3/2014

Java 9
9/2017

Java 10
3/2018

Java 11
9/2018

one feature release every six months
Milestones toward containers

Java 8  Java 10  Java 12  Java 14


Java 9  Java 11  Java 13

one feature release every six months
Milestones toward containers


One feature release every six months.
Milestones toward containers

- Large Enterprise Adaptation
- Container CPU and heap fix backported to Java 8u131 and again to 8u191

Java 8
1/1996
3/2014
Java 9

Java 10
9/2017
3/2018
Java 11

Java 12
9/2018
3/2019
Java 13

Java 14
9/2019
3/2020
Java 15
9/2020

One feature release every six months.
Milestones toward containers

- JEP 310 Application class-data sharing
- JEP 317 Experimental Java-based JIT compiler (Graal)
- JDK-8146115 Improve docker container detection and resource configuration usage
- JDK-8186348, JDK-8186309 Allow more flexibility in selecting Heap % of available RAM
- JDK-8179498 attach in linux should be relative to /proc/pid/root and namespace aware

Java 8
1/1996
3/2014

Java 9
3/2014
3/2016

Java 10
9/2017
3/2018

Java 11
9/2018
3/2019

Java 12
9/2019
3/2020

Java 13
3/2020

Java 14
9/2020

Java 15
one feature release every six months

https://medium.com/@niravksh/docker-support-in-java-10-85f9f5a52c47
Milestones toward containers

- **Java 8**
  - Large Enterprise Adoption
  - Container CPU and heap fix backported to Java 8u131 and again to 8u191

- **Java 9**
  - Module System
  - Linker, jlink
  - Ahead-of-time compilation (AOT)
  - GraalVM

- **Java 10**
- **Java 11**
- **Java 12**
- **Java 13**
- **Java 14**
- **Java 15**

- JEP 310: Application class-data sharing
- JEP 317: Experimental Java-based JIT compiler (Graal)
- JDK-8146115: Improve docker container detection and resource configuration usage
- JDK-8185248, JDK-8186109: Allow more flexibility in selecting Heap % of available RAM
- JDK-8179498: Attach in Linux should be relative to /proc/pid/root and namespace aware

- Timeline:
  - 1/1996
  - 3/2014
  - 9/2017
  - 3/2018
  - 9/2018
  - 3/2019
  - 9/2019
  - 3/2020
  - 9/2020

- One feature release every six months

References:
- [https://medium.com/apache/containers/docker-support-in-java-10-8a5b3f6ca182](https://medium.com/apache/containers/docker-support-in-java-10-8a5b3f6ca182)
Milestones toward containers

- JEP 310 Application class-data sharing
- JEP 317 Experimental Java-based JIT compiler (Graal)
- JDK-8146115 Improve docker container detection and resource configuration usage
- JDK-8186248, JDK-8186300 Allow more flexibility in selecting Heap % of available RAM
- JDK-8179498 attach in linux should be relative to /proc/pid/root and namespace aware
- JEP 189 Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)
- JEP 341 Default Class Data Sharing (CDS) Archives
- JEP 346 Promptly Return Unused Committed Memory from G1

Java 8
- Large Enterprise Adaptation
- Container CPU and heap fix backported to Java 8u131 and again to 8u191

Java 9
- Module System
- Linker, jlink
- Ahead-of-time-compilation (AOT)
- GraalVM

Java 10
- OpenJDK / GNU
- JEP 320 Remove the Java EE and CORBA Modules
- JEP 332 Transport Layer Security (TLS) 1.3

Java 12
- JEP 351 ZGC: Uncommit Unused Memory
- JEP 353 Reimplement the Legacy Socket API

Java 14
- JEP 354

Java 15


One feature release every six months.

https://medium.com/apenwarketh/docker-support-in-java-10-bbf0f8c2a227
Jlink inspects app dependencies and generates new, custom JRE with bare minimum.

Your classes, jar, or war are loaded as normal.

It's still Java JRE, just all unused modules omitted.

This new JRE loaded into your container.

For many microservices and high cohesion containers, an ideal distillation technique.

```
jlink \n   --module-path $JAVA_HOME/jmods:mlib \n   --add-modules com.greetings \n   --output greetingsapp
```

http://www.progdoc.de/papers/intSwing/t8nNotepad.jar
https://docs.oracle.com/javase/9/docs/technotes/tools/windows/jdeps.html
**Jlink** inspects app dependencies and generates new, custom JRE with bare minimum.

Your classes, jar, or war are loaded as normal.

It's still Java JRE, just all unused modules omitted.

This new JRE loaded into your container.

For many microservices and high cohesion containers, an ideal distillation technique.

```bash
jlink \
  --module-path $JAVA_HOME/jmods:mlib \
  --add-modules com.greetings \
  --output greetingsapp
```

- New, customer JRE generated in greetingsapp dir
- jdeps utility added to Java 8
- jlink utility added to Java 9
- Part of modular Java scheme

http://www.progdoc.de/papers/intSwing/t8nNotepad.jar
https://docs.oracle.com/javase/9/docs/technotes/tools/windows/jdeps.html
Java Microframeworks

Minimalistic web application frameworks

Features often provided:

- Tooling
- Libraries
- Observability *
- Debugging *
- Connectivity *
- APIs: REST, gRPC, QueryML

* Although a mesh can also help
App Class-Data Sharing

Redundant work and information is cached and shared

What if you gave each runner full moment *before* the starting gun?
App Class-Data Sharing

Redundant work and information is cached and shared

What if you gave each runner full moment before the starting gun?

Given a class name, JVM must:

- look class up on disk,
- load class,
- verify bytecode,
- pull class into internal data structure.

Multiply by hundreds or thousands of classes.
App Class-Data Sharing

Redundant work and information is cached and shared

What if you gave each runner full moment before the starting gun?

Given a class name, JVM must:
- look class up on disk,
- load class,
- verify bytecode,
- pull class into internal data structure.

Multiply by hundreds or thousands of classes.

AppCDS creates a class-data archive once and then shares it, so JVM need not recreate it.

Decreases startup time 20-50%

$ time java -Xshare:off -jar app.jar
> Hello, application class-data sharing!
> 
> real   0m0.078s
> user   0m0.094s
> sys    0m0.012s

$ time java -jar app.jar
> Hello, application class-data sharing!
> 
> real   0m0.043s
> user   0m0.053s
> sys    0m0.014s

Java 10, JEP 310: Application Class-Data Sharing
https://blog.codefx.org/java/application-class-data-sharing/#Application-Class-Data-Sharing-In-A-Nutshell
Ahead of Time Compilation (AOT)

Improved start up and runtime performance.

Where opportunities exist, code compiled to native code.

Compiled code is called by runtime engine.

Similar to Just in Time (JIT) yet JIT does little to help startup time.

Graal has a AOT compiler.

Java 9 experimental feature, JEP-295
• Developed by Oracle
• Replacement for Hotspot
• Non JVM languages: Node.js, Ruby, R, Python
• Implementations of:
  Class Data Sharing (CDS)
  Ahead of Time Compilation (AOT)
• Developed by Oracle
• Replacement for Hotspot
• Non JVM languages: Node.js, Ruby, R, Python
• Implementations of:
  Class Data Sharing (CDS)
  Ahead of Time Compilation (AOT)

Still write once run anywhere, because it’s in a container

Linker creates native image binary!
Developed by Oracle
Replacement for Hotspot
Non JVM languages: Node.js, Ruby, R, Python
Implementations of:
  Class Data Sharing (CDS)
  Ahead of Time Compilation (AOT)

Linker creates native image binary!

Still write once run anywhere, because it’s in a container
A subculture of frameworks around GraalVM wateringhole.
GraalVM™

Native images significantly reduce runtime memory requirements compared to HotSpot.

Java Microservice: Startup Time

Helidon: 15 ms → 188 ms
Micronaut: 17 ms → 2101 ms
Quarkus: 16 ms → 940 ms

~50x faster

Java Microservice: Memory Footprint

Helidon: 31 MB → 106 MB
Micronaut: 41 MB → 180 MB
Quarkus: 17 MB → 121 MB

~5x lower

http://www.graalvm.org/docs/why-graal
Quarkus

Container-first approach for cloud-native Java applications

- Make Java leading platform in Kubernetes and serverless environments.
- Programming model to address wider range of distributed application architectures.

Intended by Redhat in March 2019

---

### Memory and Boot + First Response Time

<table>
<thead>
<tr>
<th></th>
<th>Memory (REST) in Megabytes</th>
<th>Boot + First Response Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td>Quarkus + GraalVM 13 MB</td>
<td>REST + JPA 140 MB 35 MB</td>
</tr>
<tr>
<td></td>
<td>Traditional Cloud Native Stack 140 MB</td>
<td>Traditional Cloud Native Stack 218 MB</td>
</tr>
<tr>
<td>REST + JPA</td>
<td>Quarkus + GraalVM 74 MB</td>
<td>REST + JPA 130 MB 35 MB</td>
</tr>
<tr>
<td></td>
<td>Traditional Cloud Native Stack 140 MB</td>
<td>Traditional Cloud Native Stack 218 MB</td>
</tr>
</tbody>
</table>

---

https://quarkus.io/
"Every main method deserves picocli!"

- Aims to be easiest way to create rich command line applications.
- Run on and off the JVM.

Parameter parsing
Help generation
Type conversions
Subcommands
Error handling
Fast startup with Graal native compile
Distilling Java Containers
Java to Container

How hard could it be?

Dockerfile

Linux

openjdk-8_linux-x64_bin.tar.gz

HelloWorld.class
Java to Container

How hard could it be?

Dockerfile

Linux

openjdk-8_linux-x64_bin.tar.gz

HelloWorld.class

Declare

Add payload

Build

Tag

Push

Run
JVM Ergonomics

Self tuning of JVM switches to optimal defaults based on context of underlying target resources and backpressures.

- CPU and Heap usage defaults
- Garbage collector type and defaults
- Added in Java 5

- Container Aware Java
- Goldilocks settings based on container resources
- Past misbehavior with early Java 8: CPU and Heap were not limited to container
- Java switches added, helps for containers: -XX:{ Initial | Min | Max | Max } RAMPercentage
Base Containers
Some of many choices

Open Projects

- Alpine
- Portola
- Debian versions based on Toy story characters
  Stretch, Jessie, Buster, Bullseye, Bookwork, Sid
- Scratch
- Distroless: minimal Linux, glibc-based system

Bake your own base
Don’t you have enough to do? Security, performance, patches… oh my

Notable Freemiums, Frenemies

- Minimal Ubuntu, 29MB
- Universal Base Image (UBI)
- zulu-openjdk

https://hub.docker.com/_/ubuntu/
https://ubuntu.com/blog/minimal-ubuntu-released
C standard library, libc

libc:

- Library of types, functions, macros and operating systems services with resource management.
- CPU, Memory, I/O
- Language ports link to it.
**C standard library, libc**

**libc:**

- Library of types, functions, macros and operating systems services with resource management.
- CPU, Memory, I/O
- Language ports link to it.

~ dozen implementation, but two you will see often:

- GNU C Library (glibc)
  - Used in Debian and other common distros
  - Considered more complete, yet perhaps "bloated"

- musl libc
  - Several Linux distributions including Alpine Linux
  - Partially binary compatible with glibc
  - Lightweight, optimized for embedded
C standard library, libc

libc:

- Library of types, functions, macros and operating systems services with resource management.
- CPU, Memory, I/O
- Language ports link to it.

~ dozen implementation, but two you will see often:

- Used in Debian and other common distros
- Considered more complete, yet perhaps "bloated"

- Several Linux distributions including Alpine Linux
- Partially binary compatible with glibc
- Lightweight, optimized for embedded

Langs, like Java, port and validate on different OSs with libc variants.

Your app
JRE
C++
libc
OS

GNU C Library (glibc)

musl libc
5MB, without Kernel
Security-oriented
Lightweight
musl libc and busybox
Popular because of size
5MB, without Kernel
Security-oriented
Lightweight
must libc and busybox
Popular because of size

OpenJDK not supported Alpine
after 8u212-jre-alpine3.9, but has
reappeared in 13-alpine3.9.
AdoptOpenJDK
Might be a good choice to consider

- Free, open
- Built from OpenJDK
- Community maintained
- Backporting patches are community-led
- Supporting Java in containers
- For Alpine Linux they add a full glibc package made for Alpine Linux, and it replaces musl. Somewhat odd, and increases size of Alpine.

Containers images:

- latest (Full JDK)
- slim (Stripped down JDK)
- alpine (Full JDK)
- alpine-slim (Stripped down JDK)
Portola
OpenJDK Port to Alpine

Try it out...

https://www.katacoda.com/courses/docker/playground

docker run -it --rm linead/portola jshell
System.out.println("Hello")
Portola
OpenJDK Port to Alpine

- Sub project of OpenJDK
- Port to Alpine and support musl C library.
- Alpine Linux is ~4.5MB, very small
- glibc vs musl issue

Try it out...

https://www.katacoda.com/courses/docker/playground

docker run -it --rm linead/portola jshell
System.out.println("Hello")
Freemium

- Zing ($) product is a JVM tuned for performance
- "A Better JVM"
- High performance, mostly for financial systems

Zulu

- OpenJDK 15EA, 14EA, 13, 11, 8, and 7.
- Community Edition, free version
- Enterprise Edition, commercial support
- Implementation for Class Data Sharing (CDS)

Zulu Base Containers
- azul/zulu-openjdk-alpine
- azul/zulu-openjdk-centos
- azul/zulu-openjdk-debian
- azul/zulu-openjdk
Domain specific language for Dockerfiles

- Dockerfile is text source code
- Infrastructure-as-code defines container contents
- Comment lines start with #
- Other lines are: INSTRUCTION arguments
- The instruction set is small
- Instructions are ALL CAPS
- Code must start with FROM instruction

Two line Dockerfile and you have a web server

```
FROM nginx
COPY mysite /usr/share/nginx/html
```
Full Payloads with Few Lines

[Image: A sheet of paper with a post-it note saying "Was!" and another note saying "Infrastructure as code (IaC) and GitOps"]
Full Payloads with Few Lines

FROM nginx
ADD html /usr/share/nginx/html/

Infrastructure as code (IaC) and GitOps
Full Payloads with Few Lines

FROM nginx
ADD html /usr/share/nginx/html/

FROM postgres:9.5.3
ADD data /data/
RUN chmod -R 777 /data
ADD sql /docker-entrypoint-initdb.d/
FROM nginx
ADD html /usr/share/nginx/html/

FROM postgres:9.5.3
ADD data /data/
RUN chmod -R 777 /data
ADD sql /docker-entrypoint-initdb.d/

FROM java:openjdk-8-jre-alpine
ADD processor.jar processor.jar
ENTRYPOINT ["java", "-Dserver.port=9001", "-jar", "processor.jar"]
EXPOSE 9001
Multi-stage Dockerfile

New to Docker 17.05
May 2017

Provide capabilities for developers to produce sleek images without the cruft of the intermediate build artifacts.
- Moby #31067

FROM golang:1.7.3 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
Top 12 app components in containers

- Java
- Redis
- Etcd
- Nginx
- PostgreSQL
- Fluentd
- Apache
- Elastic
- MongoDB
- Php
- RabbitMQ
- Tomcat

Sysdig


An inside look at shifting container usage trends.

Second annual Docker Usage Report shows densities increasing, growing diversity in container runtimes.

___
Top 12 app components in containers

- Java
- Redis
- Etcd
- Nginx
- PostgreSQL
- Fluentd
- Apache
- Elastic
- MongoDB
- Php
- RabbitMQ
- Tomcat

Sysdig

An inside look at shifting container usage trends.
Second annual Docker Usage Report shows densities increasing, growing diversity in container runtimes.

Or look at most popular

https://hub.docker.com/search?q=&type=image
Shrinky Dinks
Shrinky Dinks

Tool: **Dive**

- Explores container images
- Explores layer contents
- Shrink image size
- Docker and OCI images
- CLI for pipelines or UI

https://github.com/wagoodman/dive
It's Just a Tar of Tars

Home of the Original **Muffulettà**
Central Grocery and Deli
New Orleans, Louisiana
It's Just a Tar of Tars

```
$ docker pull redis
Using default tag: latest
latest: Pulling from library/redis
8d691f585fa8: Pull complete
8cc0d2d17190: Pull complete
4719eb1815f2: Pull complete
200531706a7d: Pull complete
edd7c26915cf: Pull complete
e1285fcc6a46: Pull complete
Digest:
sha256:fe80393a67c7058590ca6b6903f64e35b50fa411b0496f04a85c526fb5bd2d2
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

```
$ docker save --output redis.tar redis
```
It's Just a Tar of Tars

$ docker pull redis
Using default tag: latest
latest: Pulling from library/redis
8d691f585fa8: Pull complete
8cc0d2d17190: Pull complete
4719eb1815f2: Pull complete
200531706a7d: Pull complete
eed7c26916cf: Pull complete
e1285f6c6a46: Pull complete
Digest:
sha256:fe80393a67c7058590ca6b6903f64e35b50fa411b0496f604a85c526fb5bd2d2
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest

$ docker save --output redis.tar redis

$ tar -tf redis.tar
1167eb15dc99e9c437519ce5d5fcfc46faa776b5be1b1acb86636b0c434d266eecc/
   VERSION
   json
   layer.tar
3084577582f46c3659e22d20aa16607803299745d87d4f4d36bc9a632bbcdaecc/
  799674d5f31766be4c5b95433657c01959c242160a38cfc6ccdbd5af46cf31f2/
  bad49e556e6f6ad2311dc9217b79408eb4cd61b56f9db22ad6b96179241525/
  f9c316ad0ebd0d22843e76db519a1fc385b305dc908fd7c707125e1e458e3/
  fd359d877a559df140b3480605e4e55d0aa245ad2d6a6ba4f1af70b62fd7549e/
  manifest.json
  repositories

Home of the Original Muffuletta
Central Grocery and Deli
New Orleans, Louisiana
Container Building

Many way to build a container

Gradle / Maven plugins
- Jib, Google maven and Gradle
- Ben Muschos Gradle plugin
- Fabric 8/docker-maven
- Palantir/gradle-docker

Container Build Tools
- Kaniko
- Knative
- Buildah with Podman
- Tekton
- Orca build
- Img

Running Containers without Docker using Interactive Browser-Based Scenarios

https://www.katacoda.com/courses/containers-without-docker
Explore

- Build tools
- CI/CD tools
- Registries
Container Registries

What hosts your library of private images?
Container Security Scanning

Static analysis in your pipelines

Runtime analysis on your cluster
Container Security Scanning

Static analysis in your pipelines
- Clair
- Banyan Collector
- Docker Bench for Security
- Cilium
- Anchore
- OpenSCAP Workbench
- Dagda
- Notary
- Grafaes
- Falco

Runtime analysis on your cluster
- CVE scans
- Linting and scanning
- Inspection reports
- Runtime network connectivity
- CVE scans
- SCAP and CVE
- CVE scans
- Image signing
- Custom rules, IBM Vulnerability Adviser
- Sysdig auditing tool

National Vulnerability Database
Distilling Java Containers

01 G'day
02 Distillation
03 Java
04 Containers
05 CI/CD
06 Hands On
07 Fin
Distilling Java Containers

The scenario walks through a series of steps for distilling a container to its minimum to support your Java application.

What’s the smallest image size we can achieve, with Java as the source, running on Linux?

Let’s find out...

https://katacoda.com/javajen/courses/kubernetes-fundamentals
Distilling
Java Containers
Other Katacoda Scenarios

Curious minds want to know

Learn Docker & Containers using Interactive Browser-Based Scenarios

https://developers.redhat.com/courses/

Learning Docker and Quarkus

Developing with Quarkus

4 lessons | 55 minutes

Learn Quarkus, a Kubernetes-native Java stack tailored for GraalVM & OpenJDK HotSpot. Stand up an application with a hello endpoint, then get familiar with Hibernate ORM via Panache, a Quarkus component.

https://developers.redhat.com/courses/
Additional Reads

- **Jlink**
  [https://docs.oracle.com/javase/9/tools/jlink.htm](https://docs.oracle.com/javase/9/tools/jlink.htm)

- **GraalVM**
  [https://www.graalvm.org/](https://www.graalvm.org/)

- **Multi-stage**
  [https://docs.docker.com/develop/develop-images/multistage-build/](https://docs.docker.com/develop/develop-images/multistage-build/)

- **Alpine**
  [https://en.wikipedia.org/wiki/Alpine_Linux](https://en.wikipedia.org/wiki/Alpine_Linux)

- **Portola**
  [https://openjdk.java.net/projects/portola/](https://openjdk.java.net/projects/portola/)

- **Docker JVM article**

- **Quarkus**
  [https://quarkus.io/](https://quarkus.io/)

- **Native-image**

- **Matthew Gilliard’s blog**
  [http://blog.gilliard.lol/](http://blog.gilliard.lol/)
Containers have become the preferred packaging for cloud native. Java continues to match these new targets.

Distillation Pattern · Jlink · Alpine · GraalVM · Native
Distilling
Java Containers