Java in a Container world
what we’ve done and where we’re going

Jonathan Dowland  2021-12-01
jdowland@redhat.com

Working on openjdk and containers since 2015
Presented some work at uksystems 2018
We’ll start with the history
### Java
- “Write once, run anywhere”
- Provide entire runtime environment
- JARs (& WARs, etc.): redeployable packages
- Managed runtime, Java Servlet specification, sandboxing

### Linux Containers
- **Bundle** entire runtime environment
- Open Container Initiative Image Specification
- OCI Runtime spec

There are some similarities between the goals of Java and its ecosystem and Linux containers.

Since in some cases they are both trying to solve similar problems, there can be issues getting them to work together.

I talked in 2018 about Cloud Enablement - work to containerize the Middleware product portfolio

Approx 10 products 1-2 major versions per product = 20-ish containers (then)

Perhaps the most significant output of that period was the tool cekit: pre-processor for Dockerfiles
Share build-snippets between images (mixin-style)

Since then: major version of openshift, major version of rhel, two major versions of openjdk
Resource sizing

Shell wrapper script
Configured by environment variables

```bash
java -XX:+UseParallelGC -XX:MinHeapFreeRatio=10
   -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4
   -XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError -cp "."
   -jar /deployments/spring-boot-sample-simple-1.5.0.BUILD-SNAPSHOT.jar
```

https://docs.openshift.com/container-platform/3.11/dev_guide/application_memory_sizing.html#sizing-openjdk
OpenShift Source To Image (S2I)

Openshift 3 headline feature
Produces an application image layered on top of the builder
Containing all the build tools

https://github.com/openshift/source-to-image/blob/master/docs/builder_image.md
Now
OpenJDK container limit awareness

JDK 9, 2017:
Initially requiring
-XX:+UseCGroupMemoryLimitForHeap
Backported in Red Hat OpenJDK 8

JDK 11, 2018:
Enabled by default
-X:-UseContainerSupport to turn it off

The JVM has gradually learned about container resource limits (cgroups v1 and then v2)
Initially for compatibility reasons, opt-in
Later on by default with an opt-out
Building on top of the normal S2I workflow, leveraging OpenShift buildConfigs to cherry-pick the desired build artefacts (the app) out of the S2I output, and insert it into a new image based on a slimmer, runtime-only base.

See the blog post for full details.
What time is it?
A case study

- podman run --rm $IMG ls -l /etc/localtime
  lrwxrwxrwx 1 root root 29 Oct 14 01:34 /etc/localtime -> ../usr/share/zoneinfo/Europe/London

- podman run --rm --tz Europe/Prague $IMG ls -l /etc/localtime
  -rw-r--r--. 1 root root 2338 Nov 29 17:55 /etc/localtime

A case study of a recent issue
How RHEL configures the timezone (normally): symlink, tzdata

The timezone baked into a container image might not be what you, running it, want

Podman new feature to specify the desired TZ at runtime

Implementation replaced symlink with a copy of the tzdata from host

Problem: java does not read tzdata files (internal implementation)
But it does need the symlink destination to established desirer TZ name
Podman broke this
For some domains, such as Functions-as-a-Service, initial start-up time is crucial and the JVM’s warm-up times (the initialisation of the JVM, and Hotspot, the JIT compiler, establishing hot paths, etc) are not a good fit.

There are a number of efforts taking place to address this.

Oracle Labs (a distinct part of Oracle from that responsible for Java) have a project GraalVM which implements a native-image compiler alternative to Hotspot. This relies upon a “closed-world”: all executable code known at compile time, no run-time code generation or loading.

As GraalVM is a research project, building products on top can be challenging. Mandrel is a specialised distribution of GraalVM, sponsored by red hat, that is used as a sort-of “clearing house” to provide a stable distribution of a subset of GraalVM suitable to support products built on top.

Quarkus is… a lot of things; java framework, kubernetes/openshift-supporting microservices, builds on top of Mandrel

CRIU - user-space tool for “freezing”/suspending a process to a state that can be moved between machines etc, then later “thawed”: such as a post-initialisation JVM/warm hotspot

https://developers.redhat.com/blog/2020/10/15/checkpointing-java-from-outside-of-jav
Still in early stages
Related: unfortunately named  https://openjdk.java.net/projects/crac/
https://wiki.openjdk.java.net/display/crac
Future
Cloud-native Buildpacks

In OpenShift

https://buildpacks.io/

Separate builder/runner images

Originally created by Heroku in 2011 and now adopted by Cloud Native Computing Foundation in 2018

Avoids the issue of layering the output image on top of a builder image by separating out the builder elements from a base “runtime” image into one or more “packs”

Worth a mention - ubui-micro?
Bespoke OpenJDK runtime via Java Modules

The OpenJDK distribution is quite large
Your application may not need all of it
Since JDK9 it’s modularized
Jlink, jdeps can be used to establish which java modules your application uses, and build a CUSTOM JDK with just those modules
Pilot work to integrate that into the builder processes
Additional wrinkle: system dependencies per-module need to be recorded
Thank you
jdowland@redhat.com

linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHat