AERO software stack, Java apps for cloud-native world with Quarkus

DevConf.cz
2023-06-16

Karm Babacek
Principal Software Quality Engineer

Slides: https://karms.biz/dc2k23
Overview

- **Quarkus?**
  - Best-of-breed Java libraries that you love and use. All wired on a standard backbone
  - Cloud-native as in: as small as possible, as mindful about resources as possible

- **GraalVM?**
  - Custom JDK with custom JIT, AOT capabilities, polyglot capabilities and more

- **Mandrel?**
  - native-image release specifically to support Quarkus, laser focused on Java, comes with vanilla Temurin JDK (Adoptium)

- **Native-image?**
  - A tool that compiles Java code ahead-of-time (AOT) to a binary – a native executable

- It compiles on AArch64, ship it (or not): Quality engineering
  - What we do to make the Quarkus ecosystem more and more AArch64 ready
How do I start with Quarkus on ARM?

[https://code.quarkus.io](https://code.quarkus.io)

```bash
export JAVA_HOME=/home/tester/dc2k23/jdk-17.0.7+7/;export PATH=$JAVA_HOME/bin:$PATH
unzip code-with-quarkus.zip
cd code-with-quarkus
./mvnw package -Pnative -Dquarkus.native.container-build=true
./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
```
Quarkus extensions, Quarkus platform, Quarkiverse

https://github.com/quarkusio/quarkus/tree/main/extensions

https://github.com/quarkusio/quarkus-platform

https://github.com/quarkiverse

- There are shared object (.so) libraries packaged inside many jars as various dependencies
- These libraries are either:
  - Loaded via standard Java Native Interface (JNI) at runtime in HotSpot, normal Java mode
  - Or are carefully dealt with during native-image compilation when their access to Java code is marked beforehand in a config file and their initialization is delayed to runtime, escaping the “init at build time” Quarkus native-image default
- Linux AArch64:
  - Is linux/aarch64 present?
  - How was the lib built, target ARM version etc.
Quarkus extensions, Quarkus platform, Quarkiverse

- Examples, Quarkus extensions dependencies with native libs, including Maven dev experience:
  - com/github/luben/zstd-jni (compression)
  - com/github/jnr/jffi (foreign function interface), Extension processor -> Asciidoctor -> JRuby
  - com/aayushatharva/brotli4j and brotli (compression), missing Linux aarch64 lib
  - com/swoval/file-tree-views (filesystem browsing), missing Linux aarch64 lib
  - org/fusesource/jansi (terminal control, colours etc.)
  - io/netty/netty-transport-native-epoll (I/O)
  - io/netty/incubator-netty-incubator-transport-native-io_uring (I/O, storage)
  - io/netty/netty-tcnative-boringssl-static (crypto)
  - io/grpc/grpc-netty (remote procedure call)
  - org/apache/activemq/activemq-artemis-native (messaging broker), missing Linux aarch64 lib
  - org/jetbrains/kotlin/kotlin-compiler, kotlin-daemon (Kotlin integration)
  - org/lz4/lz4-java (compression)
  - org/mongodb/mongodb-crypt (client crypto)
  - org/xerial/snappy/snappy-java (compression)
  - org/rocksdb/rocksdbjni (database)
  - org/conscrypt/conscrypt-openjdk-uber (crypto), missing Linux aarch64 lib
  - org/elasticsearch/jna (JNA for Elastic)
  - JNA (Java Native Access)
Mandrel builds, your native-image tool

- Tarballs: [https://github.com/graalvm/mandrel/releases](https://github.com/graalvm/mandrel/releases)
  - Temurin aarch64 JDK without any additional patches
  - Native-image tool from GraalVM
  - Includes AArch64

  - RHEL UBI (Universal Base Image 8, latest released)
  - As above, Temurin, native-image
  - Multiarch, amd64 + arm64
How do we test aarch64 readiness?

...small reproducers, complex apps, Quarkus, stress, perf...

- Examples:
  - Quarkus Integration Tests
    https://github.com/quarkusio/quarkus/tree/main/integration-tests
  - Mandrel Integration Tests
    https://github.com/Karm/mandrel-integration-tests
  - Selected apps from Quarkus quickstarts
  - Workbench for some targeted small apps and reproducers
    https://github.com/Karm/dev-null
  - Build scripts written in Java using JBang, anyone can build from source easily (really :))
    https://github.com/graalvm/mandrel-packaging
Multi-configuration project Linux

This build requires parameters:

REPOSITORY
Mandrel repo
https://github.com/graalvm/mandrel.git

HEADS_OR_TAGS
To be used with the repository, e.g. to use a certain head or a tag
heads

BRANCH_OR_TAG
e.g. your PR branch or a specific tag.
mandrel/23.0

PACKAGING_REPOSITORY
Mandrel packaging scripts.
https://github.com/mandrel/mandrel-packaging.git

PACKAGING_REPOSITORY_HEADS_OR_TAGS
To be used with the repository, e.g. to use a certain head or a tag

PACKAGING_REPOSITORY_BRANCH_OR_TAG
e.g. master if you use heads or some tag if you use tags.
master

MANDREL_VERSION_SUBSTRING
It must not contain spaces as it is used in tarball name too.
23.0-SNAPSHOT

MATRIX_COMBINATIONS_FILTER
Choose which combinations to run

Select: Successful - Failed - All - None

Build
HW we currently work with the most

1x CentOS 8 AArch64 – 80 cores, 256GB RAM, baremetal
1x CentOS 9 AArch64 – 80 cores, 256GB RAM, baremetal

Ampere® Altra® 64-Bit Multi-Core Processor Datasheet
Processor Subsystem
- 80 Arm® v8.2+ 64-bit CPU cores up to 3.30 GHz maximum
- 64 KB L1i-cache, 64 KB L1 D-cache per core
- 1MB L2 cache per core
- 32 MB System Level Cache (SLC)
- 2x full-width (128b) SIMD
- Coherent Mesh Interconnect (CMI):
  - Distributed snoop filtering

Source: datasheet

The architecture is aligned with the main AArch64 core of the European Processor, i.e. Neoverse N1.

Both systems are housed in a proper datacenter now :)

Runs on 2018 MediaTed Helio P70 ARMv8-A chipset just fine too.

The Quarkus application was built on the Ampere Altra machine and the executable runs on my cellphone, Cosmo Communicator.

Quarkus is ready to serve requests on its REST endpoints in 52ms.
Metrics collection

There are two main areas where we measure performance:

- **Buildtime** - how long does it take to build the application all the way from a dependency tree of all the jar files to a native executable
- **Runtime** - how well the application performs at its task, how it utilizes the resources
  - An additional runtime topic is debugging (gdb) experience

- **Metrics collector**, written with Quarkus, deployed as a native-image executable. My server does not have any Java installed :)
  [https://github.com/Karm/collector](https://github.com/Karm/collector) (Java, Quarkus, native-image, DB)
- **Dashboard**, work in progress
  [https://github.com/Karm/collector-web](https://github.com/Karm/collector-web) (C, WASM, ImPlot, Imgui)
- **Grafana** - private account, prototyping, ideas sketchbook
Buildtime metrics

The compiler: https://github.com/graalvm/mandrel/tree/mandrel/23.0/compiler

The compiler itself uses JVMCI toolchain to get processed bytecode and it itself runs on HotSpot, i.e. “normal” Java mode. Could be compiled to a native executable too though.
## Runtime metrics

- Many cores, e.g. anything that allocates resources based on available cores could affect startup
  
  ```
  -Dquarkus.vertx.event-loops-pool-size=$(EVENT_LOOPS)
  ```

- CentOS 8 vs. 9, page size 4k or 64k

- Aspiration is to gradually make the most of the architecture of the European Processor, well beyond "it runs"

The test app is giant on purpose to stress the startup and runtime.
Thank you

Quarkus was created to enable Java developers to create applications for a modern, cloud-native world. Quarkus is a Kubernetes-native Java framework tailored for GraalVM and HotSpot, crafted from best-of-breed Java libraries and standards.

Karm
https://github.com/Karm
https://twitter.com/_karm
https://www.linkedin.com/in/karmmichal

https://quarkus.io
https://twitter.com/AERO_Project_EU
https://twitter.com/QuarkusIO
https://github.com/graalvm/mandrel
https://github.com/quarkusio/quarkus
Join us for this cool session:

**Quarkus Super-Heroes Workshop**

Saturday, June 17 at 10:15am - 11:35am
Speaker: Martin Štefanko