QUARKUS

Supersonic, Subatomic, Native!

Foivos Zakkak
Senior Software Engineer
fzakkak@redhat.com

Red Hat

What we'll discuss today

- Quarkus vs Quarkus Native
- Mandrel
- Tips for developing Quarkus Native applications

Make IT 2022
Red Hat

Quarkus vs Quarkus Native

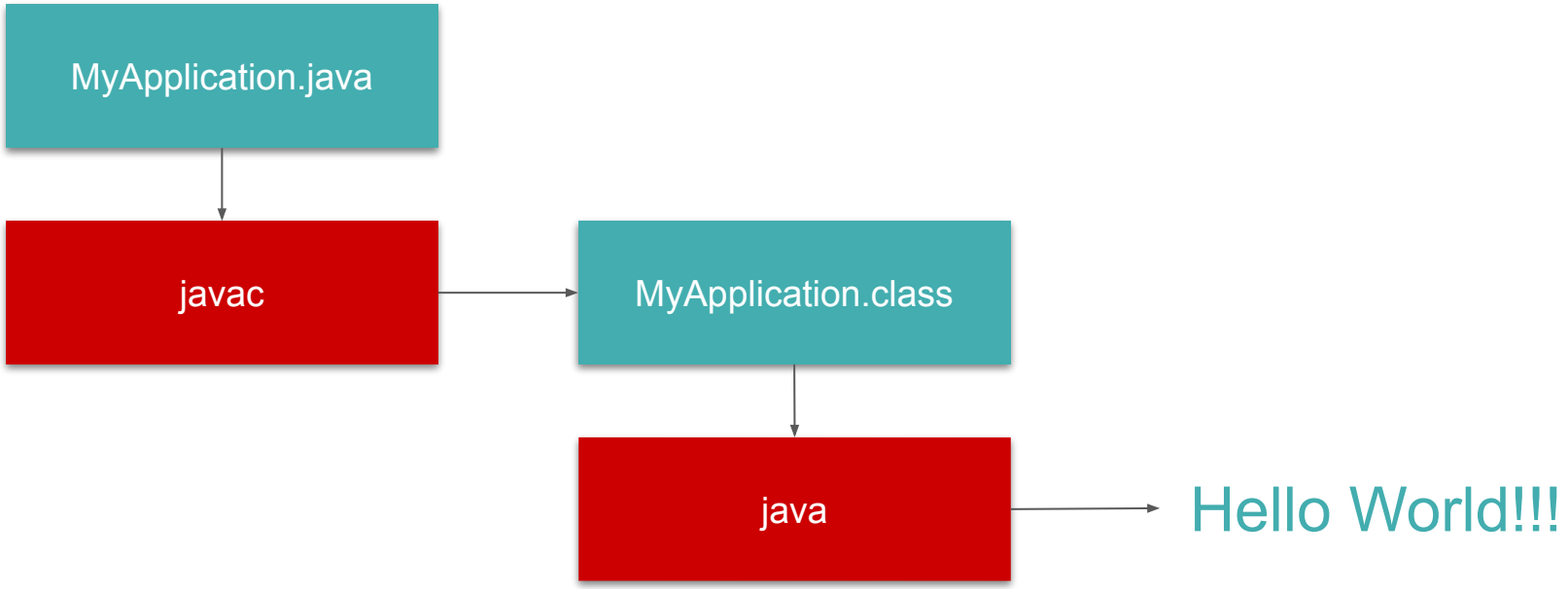
Make IT 2022

Red Hat

What is Quarkus Native?

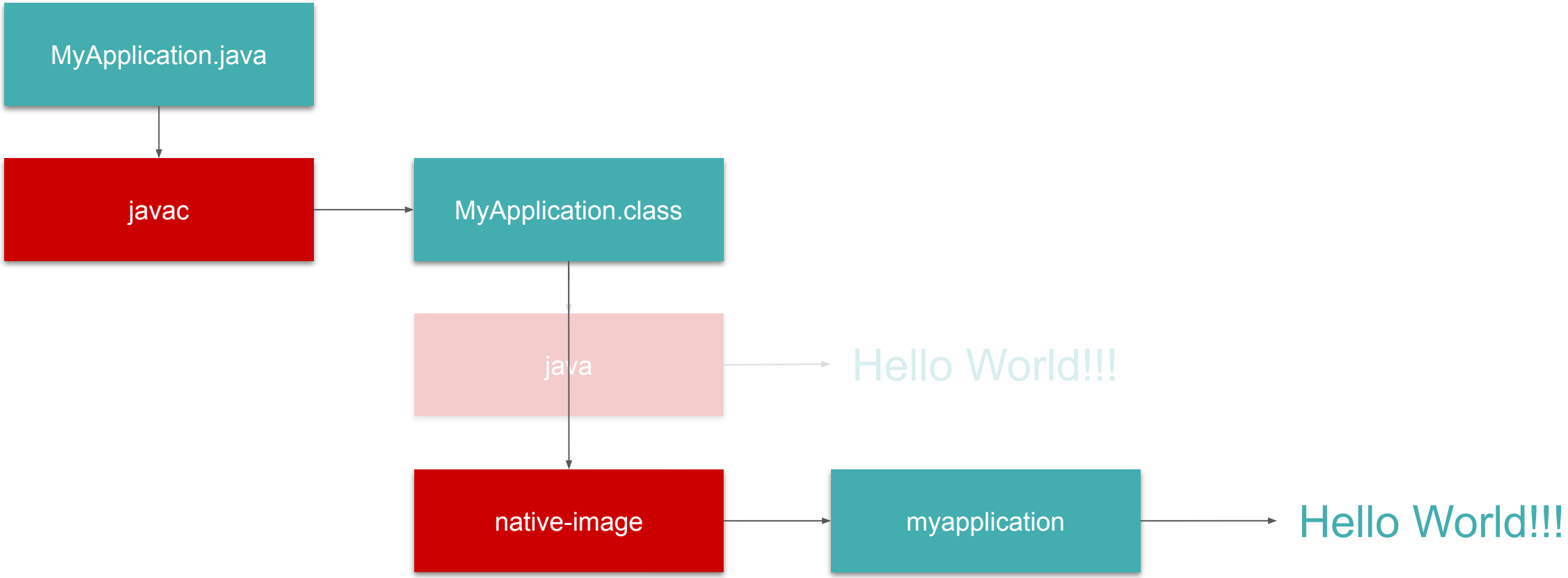
- Ahead of Time (AoT) compilation using GraalVM's native-image
- Most Quarkus extensions
  - compatible with native-image "out of the box"
  - optimized to help native-image eliminate dead code and avoid inclusion of unnecessary code/data

The standard Quarkus workflow



Make IT 2022
Red Hat

The Quarkus Native workflow



Make IT 2022
Red Hat

Quarkus vs Quarkus Native: Build Time

Quarkus

```
$ ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< org.acme:code-with-quarkus >----------------------
[INFO] Building code-with-quarkus 1.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  15.928 s
[INFO] Finished at: 2022-05-09T17:08:44+03:00
[INFO] ------------------------------------------------------------------------
```

Quarkus Native

```
$ ./mvnw -Pnative clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< org.acme:code-with-quarkus >----------------------
[INFO] Building code-with-quarkus 1.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:56 min
[INFO] Finished at: 2022-05-09T17:24:35+03:00
[INFO] ------------------------------------------------------------------------
```


Make IT 2022
Red Hat

Quarkus vs Quarkus Native: Startup Time

Quarkus

```
$ java -jar ./target/quarkus-app/quarkus-run.jar

__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2022-05-09 17:11:10,119 INFO  [io.quarkus] (main) code-with-quarkus 1.0.0-SNAPSHOT on JVM (powered by Quarkus 2.8.3.Final) started in 2.580s.
Listening on: http://0.0.0.0:8080
2022-05-09 17:11:10,173 INFO  [io.quarkus] (main) Profile prod activated.
2022-05-09 17:11:10,173 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy, smallrye-context-propagation, vertx]
```

Quarkus Native

```
$ ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner

__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2022-05-09 16:55:12,127 INFO  [io.quarkus] (main) code-with-quarkus 1.0.0-SNAPSHOT native (powered by Quarkus 2.8.3.Final) started in 0.039s.
Listening on: http://0.0.0.0:8080
2022-05-09 16:55:12,127 INFO  [io.quarkus] (main) Profile prod activated.
2022-05-09 16:55:12,127 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy, smallrye-context-propagation, vertx]
```

Quarkus vs Quarkus Native: Portability

Quarkus

```
$ file ./target/quarkus-app/quarkus-run.jar
target/quarkus-app/quarkus-run.jar: Zip archive data, at least v1.0 to extract, compression method=store
```

Quarkus Native

```
$ file ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
./target/code-with-quarkus-1.0.0-SNAPSHOT-runner: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, BuildID[sha1]=87022586211d24a49f3e47070c6cfa7c7a2e8396, for GNU/Linux 3.2.0, not stripped
```


Make IT 2022

Red Hat

Quarkus vs Quarkus Native: Application size

Quarkus

```
$ du -h target/quarkus-app/quarkus-run.jar
4.0K   target/quarkus-app/quarkus-run.jar


$ du -hs /opt/jvms/jdk-11.0.15+10
319M   /opt/jvms/jdk-11.0.15+10
```

Quarkus Native

```
$ du -h ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
39M    target/code-with-quarkus-1.0.0-SNAPSHOT-runner
```


Make IT 2022

Red Hat

Quarkus vs Quarkus Native: Container size

Quarkus

```
$ ./mvnw quarkus:add-extension -Dextensions="container-image-docker"
...

$ ./mvnw clean package -Dquarkus.container-image.build=true
...
[INFO] Total time:  48.099 s
[INFO] Finished at: 2022-05-10T15:44:58+03:00
[INFO] ------------------------------------------------------------------------

$ docker images
REPOSITORY                                                  TAG                 IMAGE ID       CREATED         SIZE
zakkak/code-with-quarkus                                    1.0.0-SNAPSHOT      80d24a5624bf   3 minutes ago   448MB
```

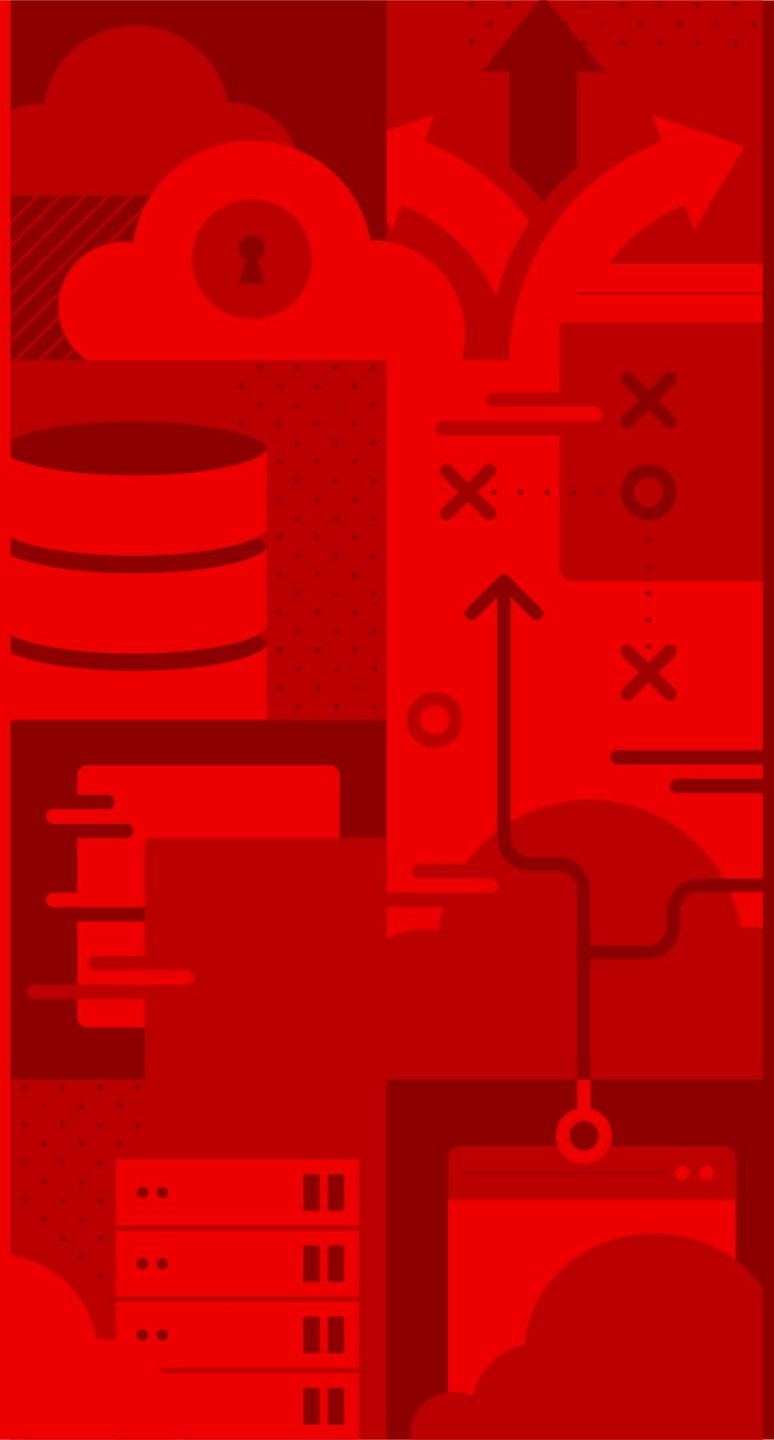
Quarkus Native

```
$ ./mvnw -Pnative clean package -Dquarkus.container-image.build=true
...
[INFO] Total time:  01:39 min
[INFO] Finished at: 2022-05-10T15:49:59+03:00
[INFO] ------------------------------------------------------------------------


$ docker images
REPOSITORY                                                  TAG                 IMAGE ID       CREATED         SIZE
zakkak/code-with-quarkus                                    1.0.0-SNAPSHOT      2e2d622b2d7c   2 minutes ago   142MB
```


Make IT 2022

Red Hat



Is Quarkus Native always better?

Quarkus Native Pros

- Fast start up times
  - No JVM start-up overhead
    - No Class loading and verification
    - Build time initialization (BTI)
- Close to peak performance from start
  - No JVM warm up needed
  - No JIT compilation
- Small standalone binary
  - Less dependencies
  - Smaller memory footprint (does it matter?)
- Smaller Resident Set Size (RSS)
  - native doesn't hold all the metadata that JVM needs at runtime
  - Heap image is shared across multiple instances (copy on write)

See https://github.com/oracle/graal/issues/979#issuecomment-480786612 for more information regarding performance differences

Make IT 2022
Red Hat

Quarkus Native Cons

- Lower peak performance compared
  - No JIT / dynamic optimizations
  - Worse GC implementation
- Slower development cycle
  - Develop and test in JVM mode
- Lacks behind in terms of tooling
  - Harder to debug and monitor
- Security patches require recompilation
  - Even if the issue is not in the application code

See https://github.com/oracle/graal/issues/979#issuecomment-480786612 for more information regarding performance differences

Make IT 2022
Red Hat

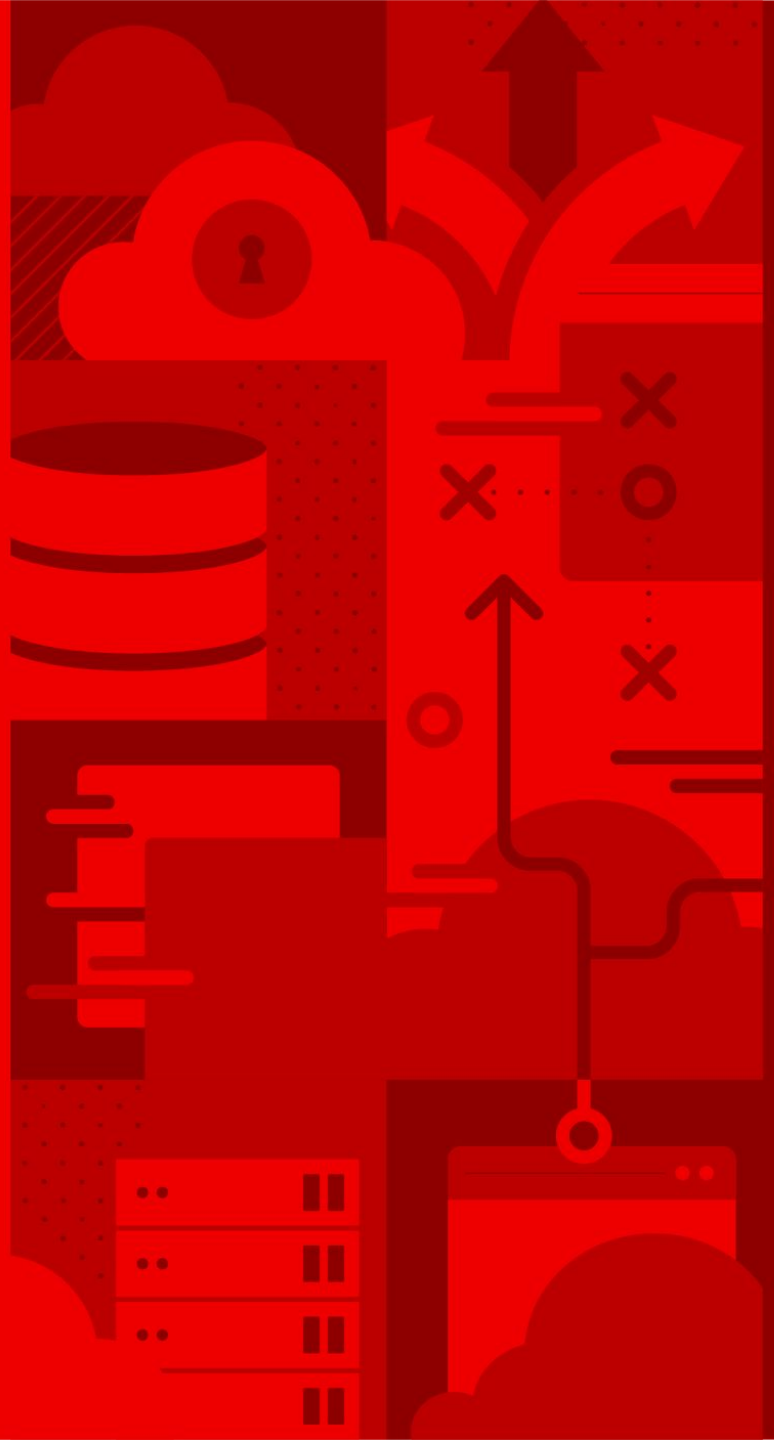
Limitations

Often require explicit configuration:

- Dynamic Class loading
- Dynamic Proxy
- Reflection
- Java Native Interface
- Serialization

- MethodHandles and invokedynamic bytecode
  - Lambdas are supported
- Tooling support
  - No JDWP, agents, JMX, JVMTI, etc.

See https://www.graalvm.org/22.1/reference-manual/native-image/Limitations/ for more information



How does it work?

How does Quarkus Native work?

- Takes advantage of GraalVM's native-image
  - Close the world assumption / analysis
    - Identify reachable code and data using static analysis
    - Only compile the reachable part, drop the rest
    - No need to compile / include code that runs at build time, e.g., Build Time Initialization
- Quarkus extensions handle the biggest part of configuration for indirectly accessed code / data
  - Allows use of "dynamic" class loading, reflection, JNI, etc.
  - Allows the embedding of resources, e.g., configuration files, in the binary
- Quarkus annotations and native-image configuration allow for further configuration

Quarkus Native Defaults

- Build time initialization of all classes (where possible)
  - Re-initialize when necessary (e.g. random seeds)
  - Reset fields to null to prevent pulling in undesired state or classes
- Doesn't allow incomplete classpaths (`--link-at-build-time`)
  - No unexpected runtime failures due to ClassNotFoundException
- Disables `ParseOnce` option of native-image
  - Reduces memory usage
  - Increases build-times in some cases
  - Pass `-Dquarkus.native.additional-build-args="-H:+ParseOnce"` to override

Mandrel:
A specialized distribution of GraalVM for Quarkus

Make IT 2022

Red Hat

High level description of Mandrel

- Downstream distribution of GraalVM CE
- Focused solely on GraalVM's native-image feature
  - No polyglot support
  - No Truffle support
  - etc.
- Specifically tailored to support Quarkus Native
- Currently supported platforms:
  - Linux AMD64 and AArch64
  - Windows AMD64

Make IT 2022
Red Hat

Development Process

New features or bug fixes are first submitted upstream on GraalVM's github repository

Once merged upstream they are considered for backporting, depending on the added value

The changes land either in the next patch release if backported or in the next feature or CPU release if not backported

Release Schedule and Maintenance

- Mandrel releases follow closely after the corresponding upstream GraalVM CE releases on a best-efforts basis
  - LTS releases get updated on a quarterly basis, following the OpenJDK CPU updates schedule
    - Always use the latest release or the latest supported LTS
  - Mandrel LTS releases do not necessarily get the same support length with GraalVM's LTS releases
    - The support's length is defined by the needs of Quarkus

Make IT 2022
Red Hat

Versioning scheme

- Mandrel releases are versioned as YY.X.Z.W
  - The YY.X.Z part denotes the corresponding GraalVM CE version
  - The W part is the Mandrel patch version

```
$ ~/.sdkman/candidates/java/22.1.0.0.r11-mandrel/bin/native-image --version
native-image 22.1.0.0-Final Mandrel Distribution (Java Version 11.0.15+10)

$ ~/.sdkman/candidates/java/22.1.0.r11-grl/bin/native-image --version
GraalVM 22.1.0 Java 11 CE (Java Version 11.0.15+10-jvmci-22.1-b06)
```

Under the hood

Built as an extension of upstream OpenJDK (11 & 17)

- Based on Eclipse Temurin releases

```
$ ~/.sdkman/candidates/java/22.1.0.0.r11-mandrel/bin/native-image --version
native-image 22.1.0.0-Final Mandrel Distribution (Java Version 11.0.15+10)

$ ~/.sdkman/candidates/java/22.1.0.0.r11-mandrel/bin/java --version
openjdk 11.0.15 2022-04-19
OpenJDK Runtime Environment Temurin-11.0.15+10 (build 11.0.15+10)
OpenJDK 64-Bit Server VM Temurin-11.0.15+10 (build 11.0.15+10, mixed mode)
```

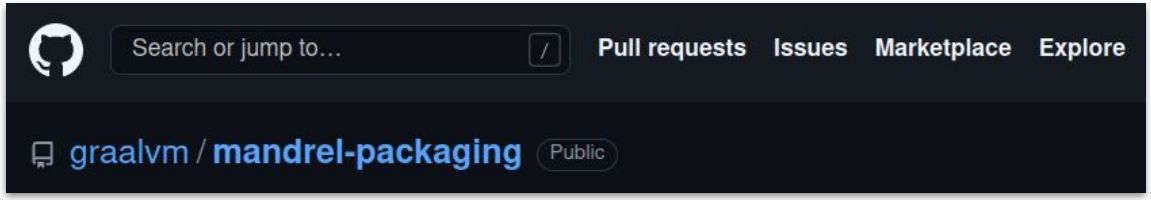
- Comprises mostly of a set of jar files and a few native libraries

Getting Mandrel

- Quarkus by default relies on pre-built mandrel images hosted on https://quay.io/repository/quarkus/ubi-quarkus-mandrel
  It will automatically fetch them for you if you tell it to use mandrel
  `-Dquarkus.native.builder-image=mandrel`
- Using SDKMAN
  $ sdk install java 22.1.0.0.r17-mandrel
- Manually through https://github.com/graalvm/mandrel/releases

Building your own

Mandrel relies on mandrel-packaging to be built



README.md

Mandrel Packaging

This repo contains all the necessary scripts and tools to build Mandrel.

Building mandrelJDK locally

```
export JAVA_HOME=/opt/jvms/openjdk-11.0.8_10
$JAVA_HOME/bin/java -ea build.java --mx-home ~/code/mx --mandrel-repo ~/code/mandrel
```

where:

- `JAVA_HOME` is the path to the OpenJDK you want to use for building mandrel.
- `--mx-home` is the path where you cloned https://github.com/graalvm/mx. Defaults to `/opt/mx`.
- `--mandrel-repo` is the path where you cloned https://github.com/graalvm/mandrel. Defaults to `/tmp/mandrel`.

Make IT 2022
Red Hat

Is this future-proof?

- native-image is an existing practical approach to Native Java
  - still comes with some gray zones
  - tries to act as much as possible like JVM
    - not always possible
- we need a clear specification for Native Java

Make IT 2022
Red Hat

Meet project Leyden

- *"Leyden will add static images to the Java Platform Specification, and we expect that GraalVM will evolve to implement that Specification."* - 27 Apr 2020, Mark Reinhold. See https://mail.openjdk.java.net/pipermail/discuss/2020-April/005429.html
- On 20 May 2022 Mark Reinhold posted "Project Leyden: Beginnings" which kick-started some discussions: *"So rather than adopt the closed-world constraint at the start, I propose that we instead pursue a gradual, incremental approach."* See https://openjdk.java.net/projects/leyden/notes/01-beginnings
- Still early to draw conclusions but there is movement towards a specification for Native Java (a.k.a. static images)

Make IT 2022
Red Hat

Developing Quarkus Native Applications

Make IT 2022

Red Hat

Developing Quarkus native applications

- Always develop and test the application in JVM mode first
- Compile to native

```
$ ./mvnw -Pnative package
```

- Compile to native using Mandrel

```
$ ./mvnw -Pnative -Dquarkus.native.builder-image=mandrel package
```


Make IT 2022
Red Hat

Resource requirements

- There are cases that require more than 4GB of memory
  - `quarkus.native.native-image-xmx` controls max heap size

    e.g. `-Dquarkus.native.native-image-xmx=5g`
- Ensure containers have enough resources
  - both memory and processors
- Building might appear like it got stack
  - Often caused by repetitive GC cycles due to low memory
    - native-image is a Java program

      it can be monitored using the usual Java tools

Make IT 2022
Red Hat

Generic hints

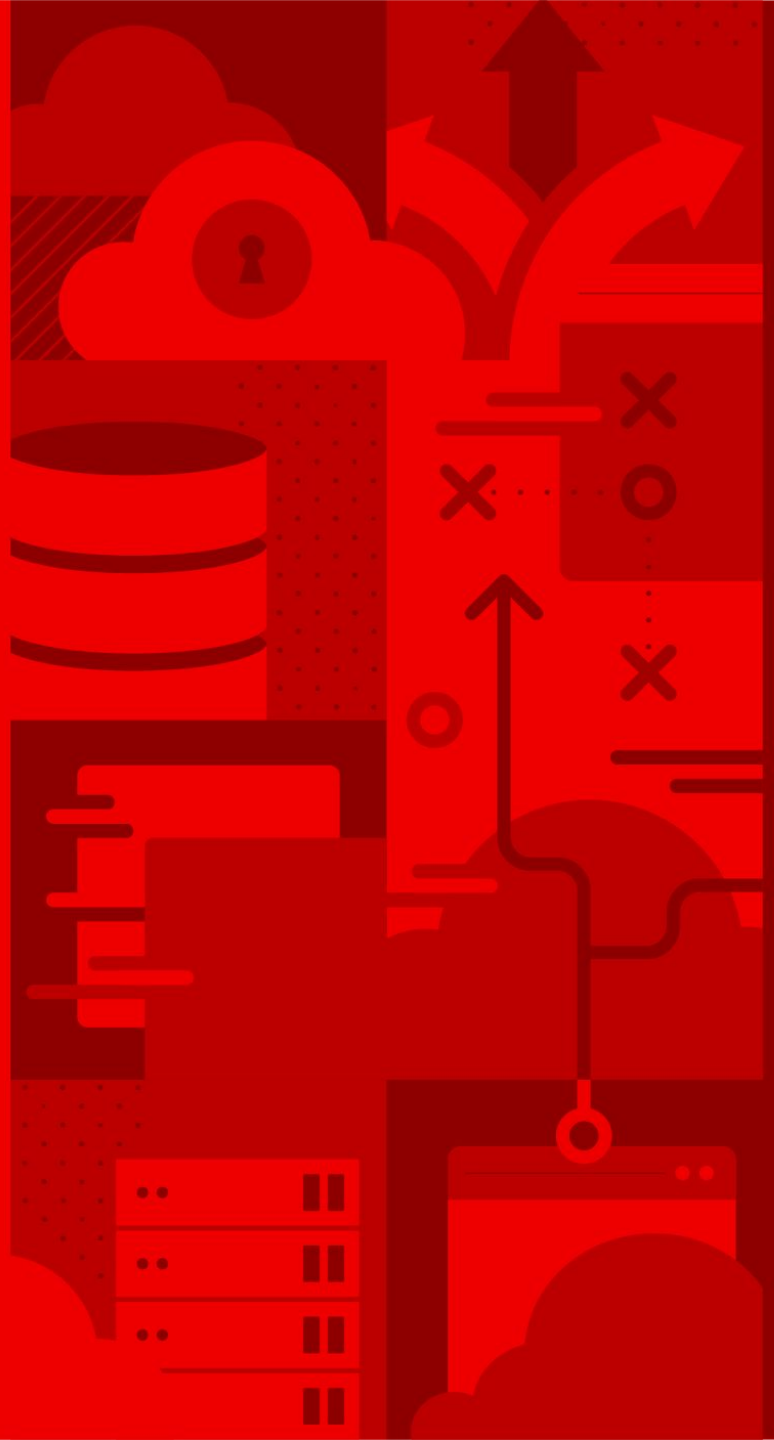
- Write native tests (see `@QuarkusIntegrationTest` annotation)
- Less reachable classes means better performance
  - Avoid pulling in unnecessary classes / libraries
  - Avoid making soft dependencies appear like hard ones
- Avoid reflection and serialization if possible
  - Use `@RegisterForReflection` to register classes if necessary
- By default Quarkus automatically includes any resources present under META-INF/resources
  - To add more use quarkus.native.resources.includes
  - To add less use quarkus.native.resources.excludes

Make IT 2022
Red Hat

Generic hints

- Read:
  - https://quarkus.io/guides/building-native-image
  - https://quarkus.io/guides/writing-native-applications-tips

Make IT 2022
Red Hat



Beware of the substitutions!

Beware of the substitutions

- Both Quarkus and GraalVM perform substitutions (patching of existing code)
  - GraalVM patches only JDK code
    (about 1500 substitutions and 229 deletions)
    - to remove HotSpot dependencies
    - for native-image compatibility
    - for performance
- Quarkus patches library code
  (about 300 substitutions and 20 deletions)
  - for native-image compatibility
  - for performance

Make IT 2022
Red Hat

Beware of the substitutions

- Some substitutions drop support for some features
  - Read the documentation

```
@TargetClass(className = “com.datical.liquibase.ext.command.init.StartH2CommandStep")
final class SubstituteStartH2CommandStep {

    @Substitute
    public void run(CommandResultsBuilder builder) {
        throw new UnsupportedOperationException(
                "Starting h2 is not supported by quarkus-liquibase");
    }

}
```


Make IT 2022
Red Hat

Beware of the substitutions

- Avoid writing your own substitutions
  - Nightmare to maintain – *Need to update every time the source changes*
  - Easy to get wrong – *Might not fully understand what you are changing*
  - Duplicated effort – *Others will probably need to do the same*
- Make the corresponding code native-image ready / friendly
  - Usually there is not much resistance from upstream
  - Not always possible (closed source 3rd party dependencies)

Make IT 2022
Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat