What's "Native Java"?

A discussion based on the Quarkus native paradigm

Foivos Zakkak
Software Engineer

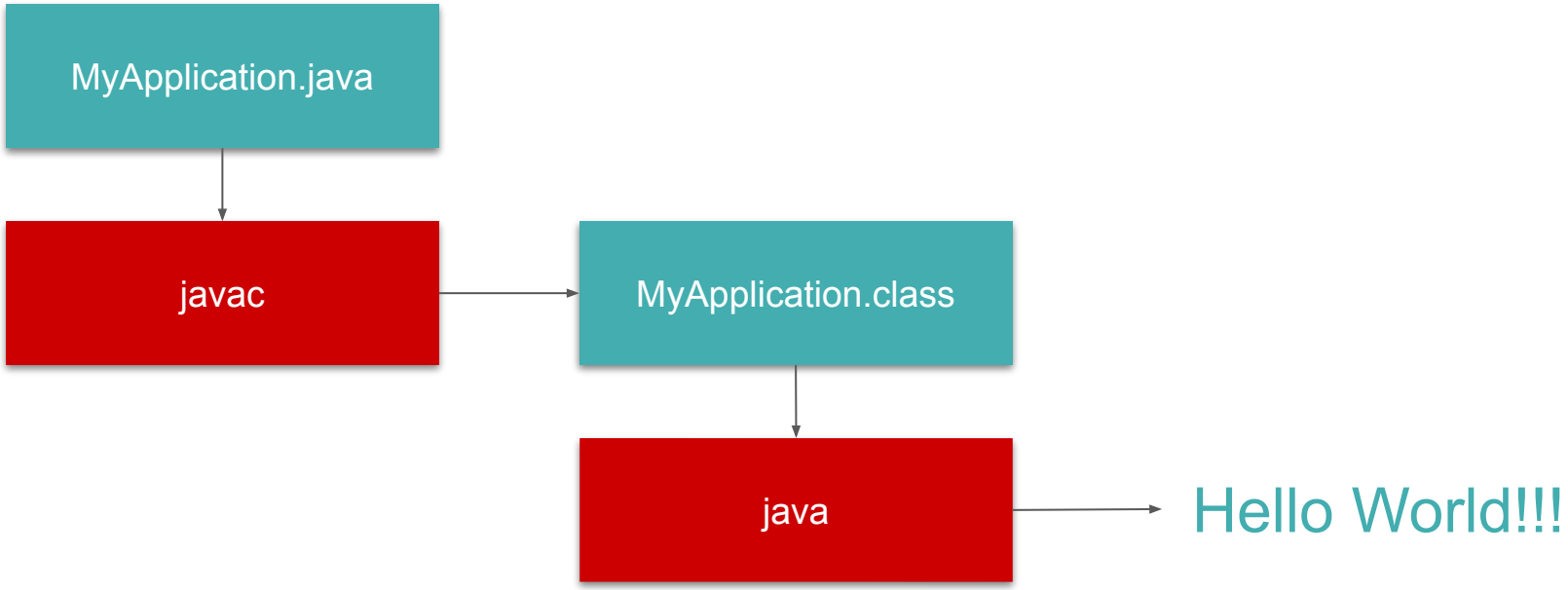
@zakkak@mastodon.online
@foivoszakkak
https://foivos.zakkak.net

Red Hat

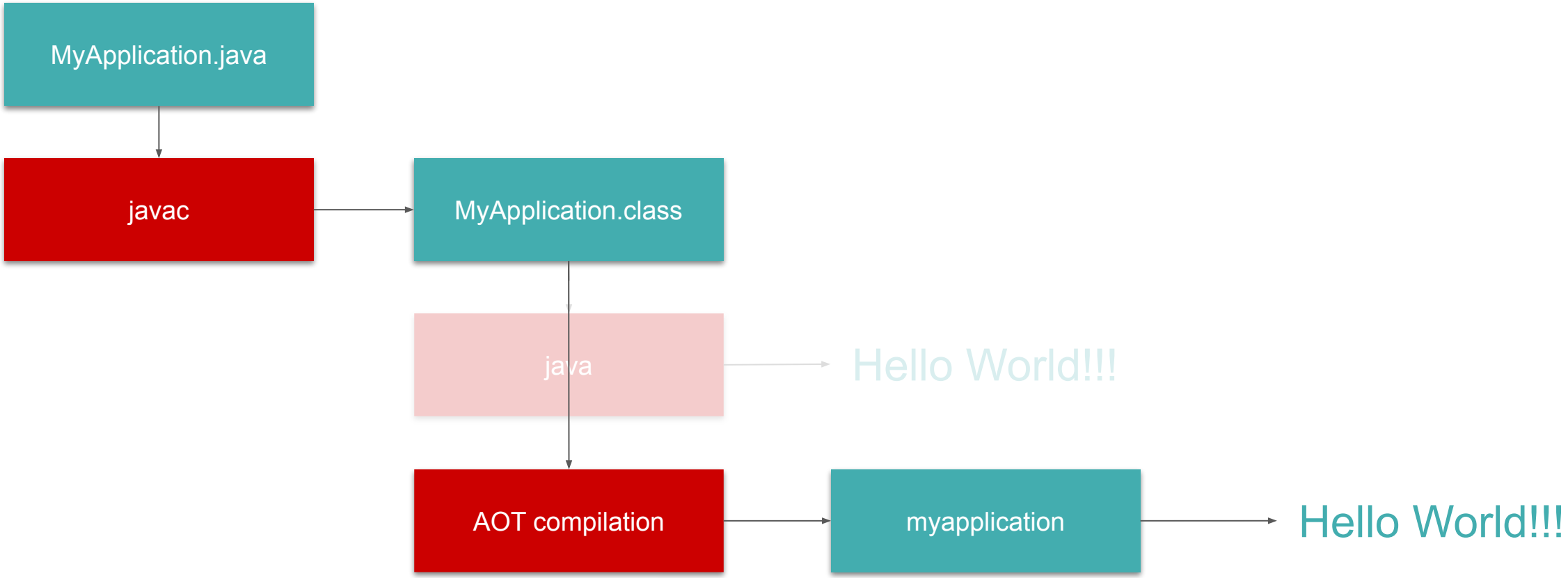
What is Quarkus Native?

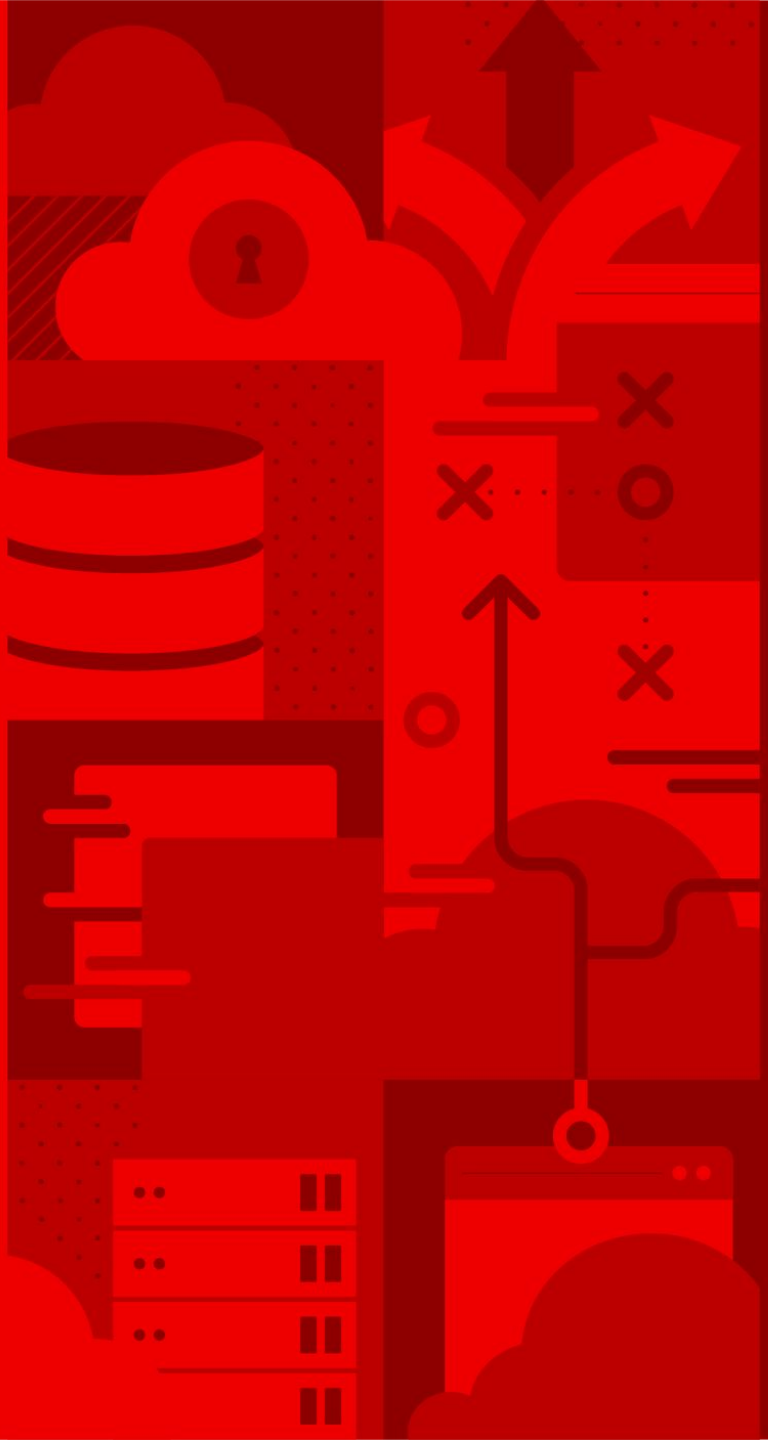
- Ahead of Time (AoT) compilation of Java to binary
- Most Quarkus extensions are:
  - compatible with native-mode “out of the box”
  - optimized to help AOT compilation eliminate dead code and avoid inclusion of unnecessary code/data

The standard Quarkus workflow



The Quarkus Native workflow





Let’s do some side by side comparisons!

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] ------------------------------------------------------------------------
```

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
```

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
```

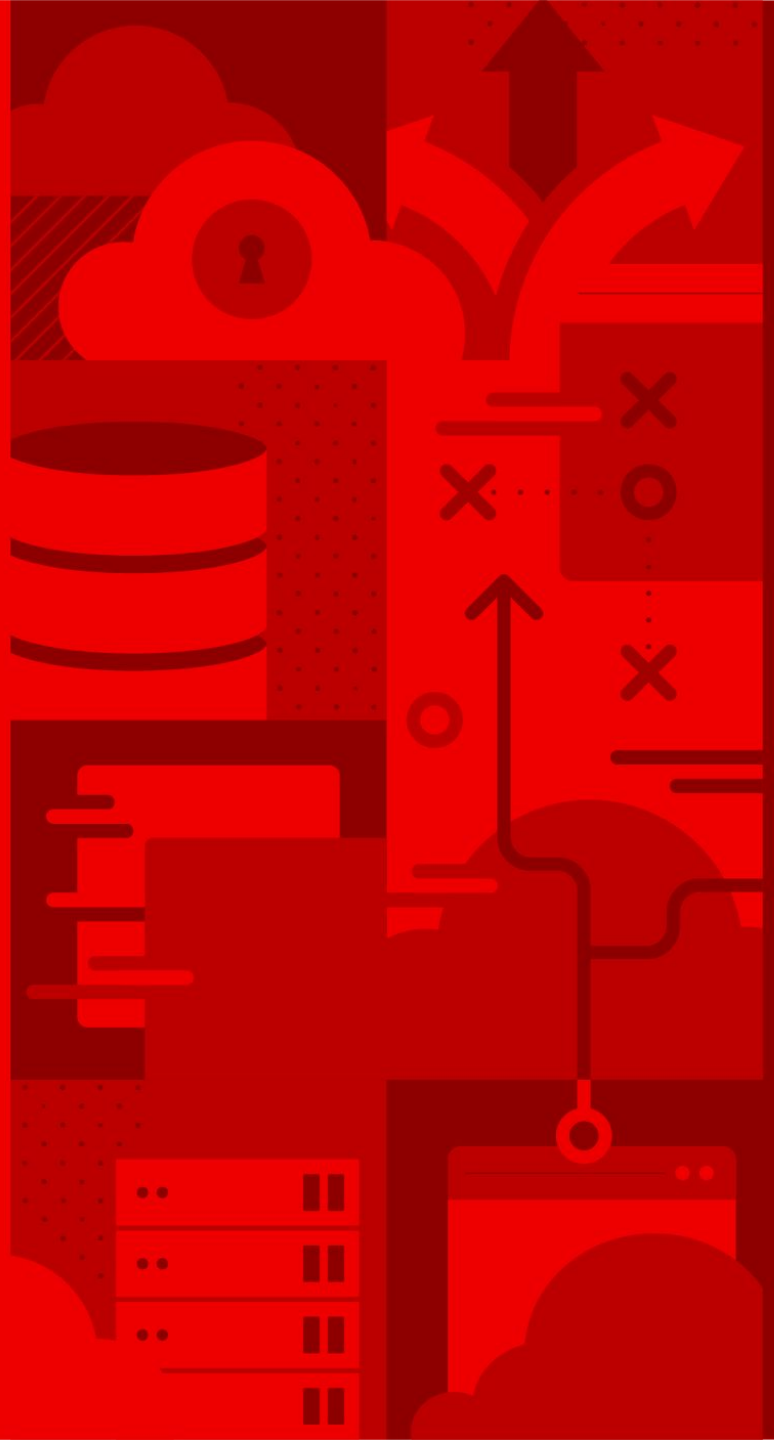
Quarkus vs Quarkus Native: Container size

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
```



Is Quarkus Native always better?

No!

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 footprint on disk (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

Quarkus Native Cons

- Lower peak performance compared to JVM mode
  - 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
- Not portable
  - Need to build different binaries for different platforms

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

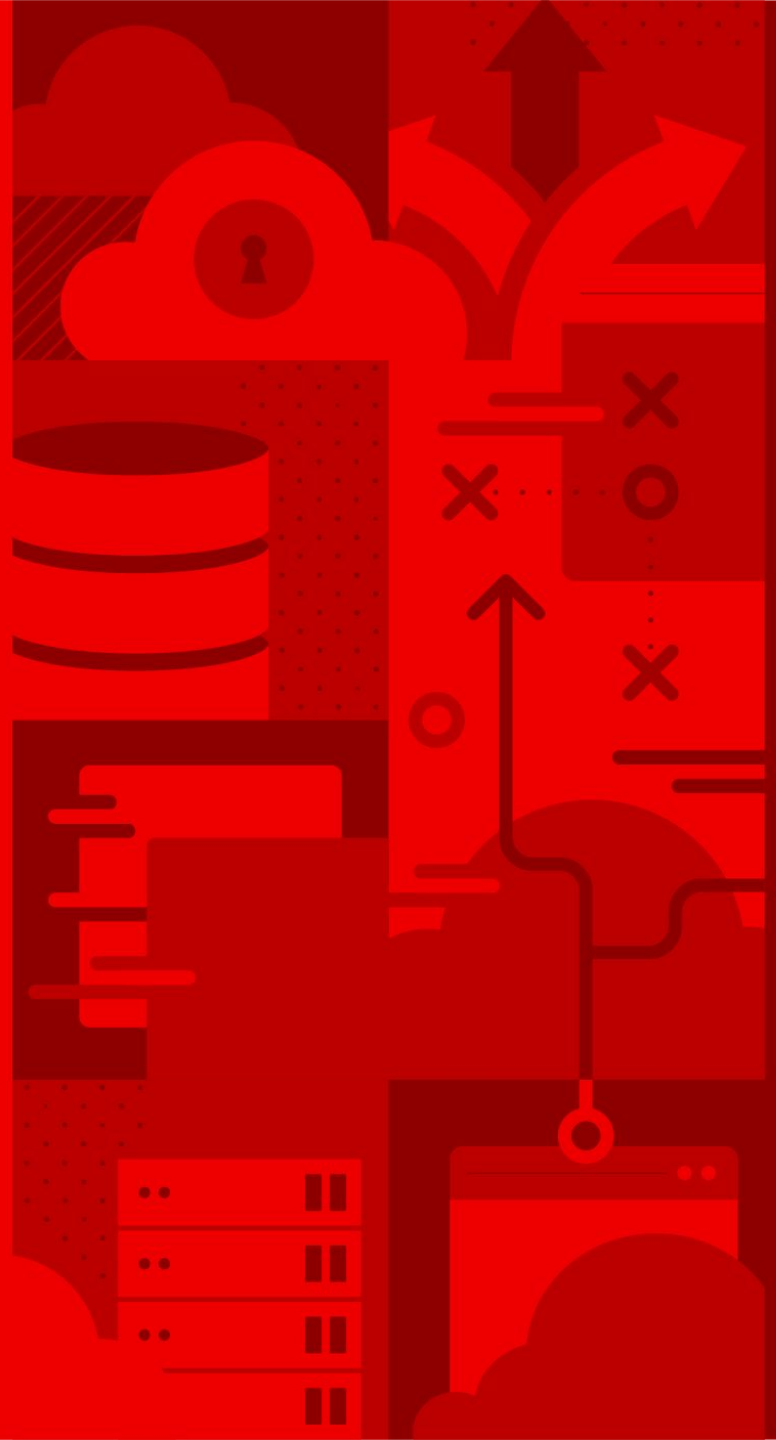
Quakrus Native Limitations

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.

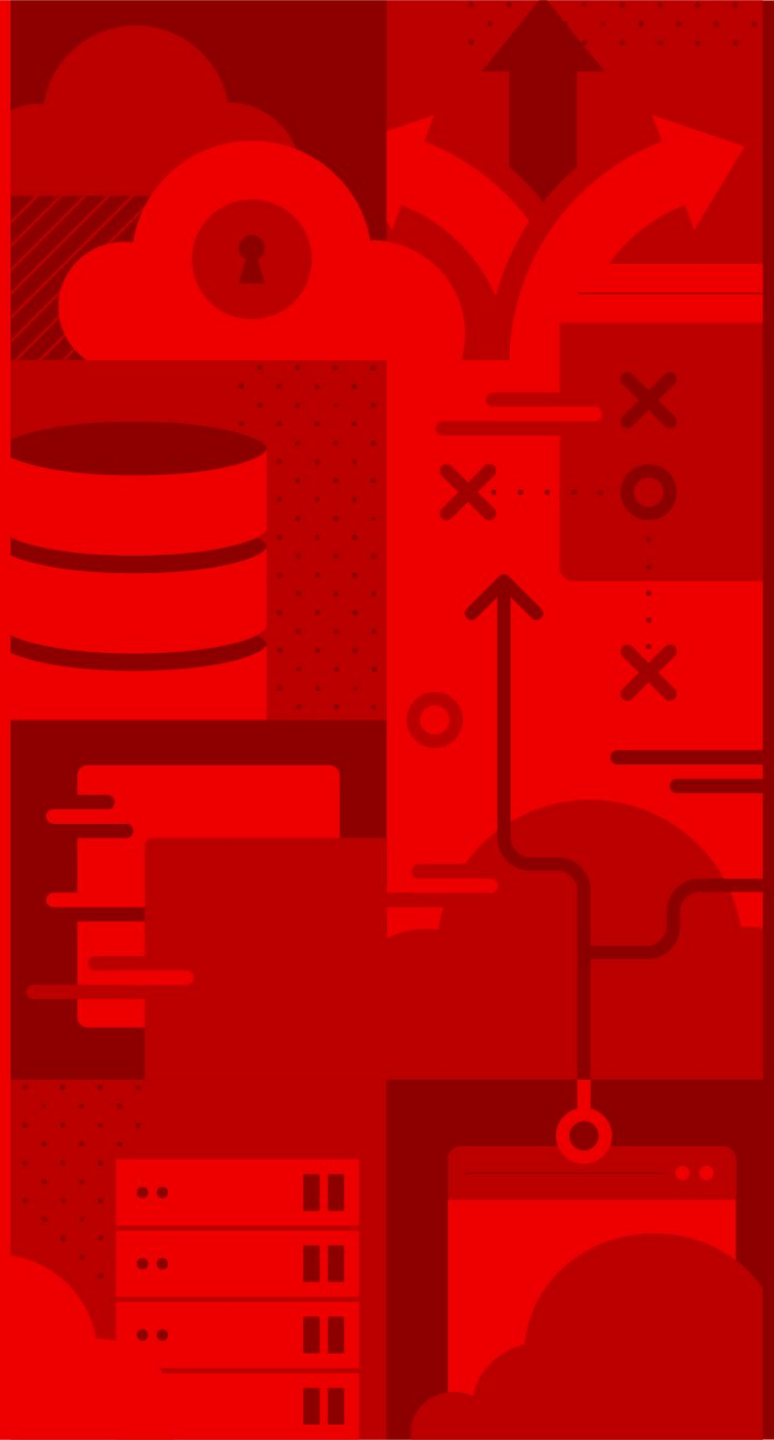
See https://www.graalvm.org/22.1/reference-manual/native-image/Limitations/ for more information



So when should one use Quarkus Native?

When to prefer Quarkus Native

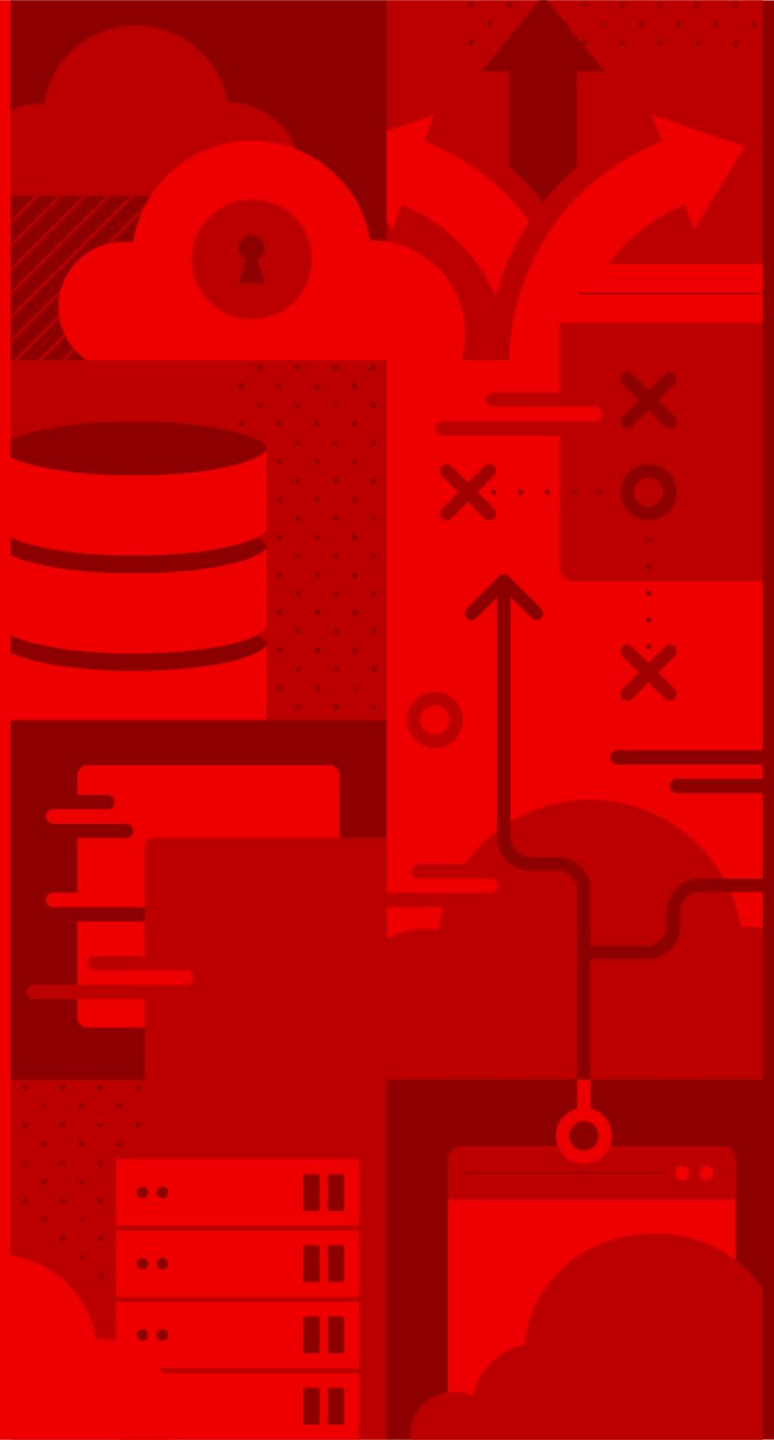
- Typically better for:
  - Short lived processes
  - Processes that require fast startup
  - Non-GC-heavy workloads
- Typically worse for:
  - Highly dynamic workloads
    (using a lot of reflection and dynamic class loading)
  - Processes that can benefit from higher peak performance
    - Usually that means they will also need to run for longer



How does it work?

How does Quarkus Native work?

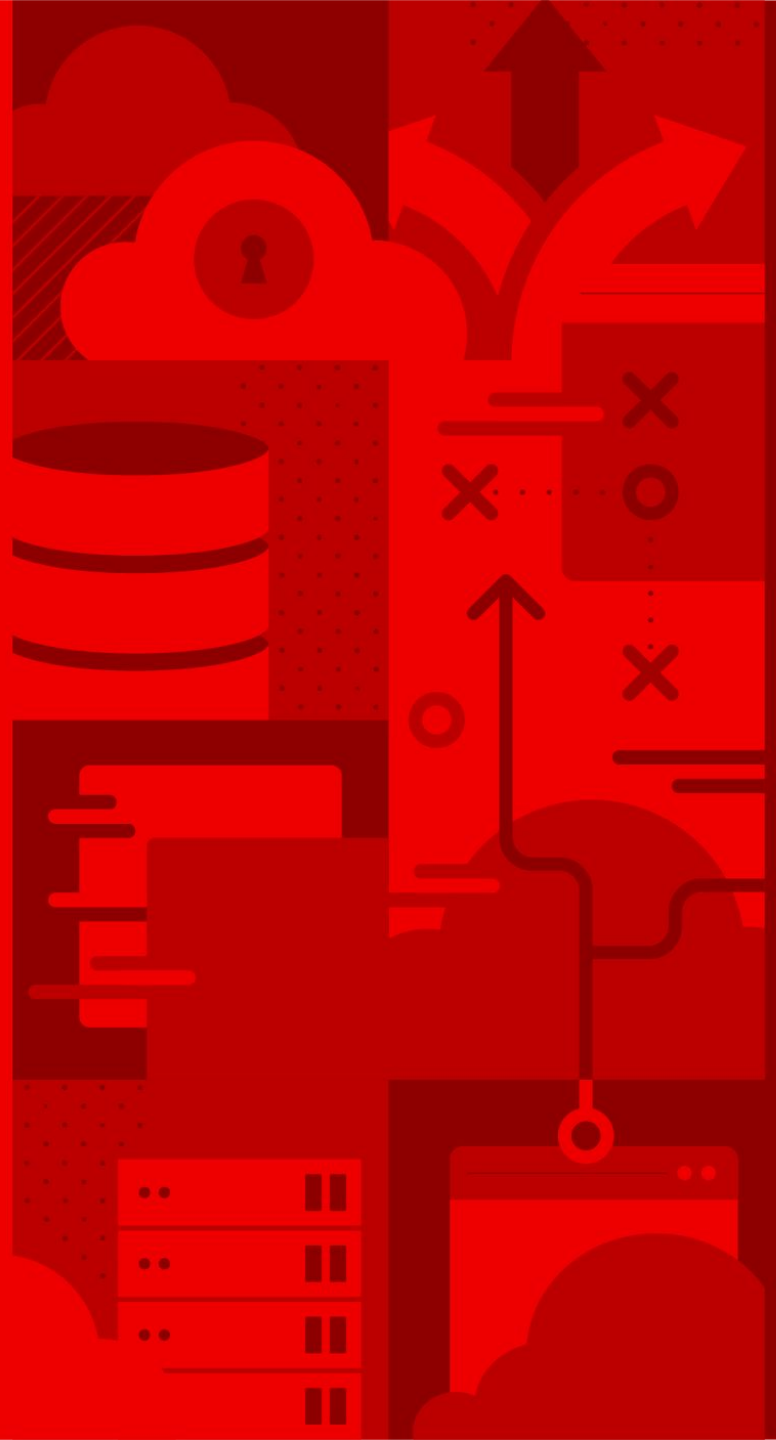
- Takes advantage of GraalVM's native-image
  - Closed world assumption / analysis
    - Identify reachable code and data using static analysis
    - Only compile the reachable part, drop the rest
  - Initialize Once, Start Fast!
    - No need to compile / include code that runs at build time, e.g., 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



How does it defer?

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
- Opinionated native-image support for libraries (due to build time initialization)
  - By default GraalVM offers a metadata repository, which Quarkus doesn't use



Is it future-proof?

Is Quarkus Native future-proof?

- native-image is an existing practical approach to Native Java
  - already widely deployed in production
  - 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

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)

https://www.infoq.com/articles/native-java-aligning/

Facilitating the Spread of Knowledge and Innovation in Professional Software Development | English Edition | Contribute

Search

InfoQ

Development | Architecture & Design | AI, ML & Data Engineering | Culture & Methods

2,359,240 May unique visitors

JUN 21 **InfoQ Live June** Learn how cloud architectures achieve cost savings, improve reliability & deliver value. Register Now.

JUL 19 **InfoQ Live July** Learn how to migrate an application to serverless and what are the common mistakes to avoid. Register Now.

OCT 24-28 **QCon San Fra** Understand the e on Oct 24-28, 202

InfoQ Homepage > Articles > Standardizing Native Java: Aligning GraalVM And OpenJDK

JAVA

Web Server and Reverse-Proxy

Standardizing Native Java: Aligning GraalVM and OpenJDK

LIKE | DISCUSS

MAY 30, 2022 • 23 MIN READ

by

Andrew Dinn FOLLOW

Distiguished Engineer in the Red Hat's Java Team

Dan Heidinga FOLLOW

Principal Software Engineer at Red Hat

AND 1 MORE

Key Takeaways

- Native Java is essential for Java to remain relevant in the evolving cloud world.
- Native Java is not a solved problem yet.
- The development lifecycle needs to adapt as well.
- Standardization through Project Leyden is key to the success of native Java.
- Native Java needs to be brought into OpenJDK to enable co-evolution with other ongoing enhancements.

RE

Java and Upc

MAY

Rev Nat

APR

devstaff Red Hat

Questions?

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