Quarkus native How hard can it be ?

Miniconf/21-04-2024

Ronald Spierenburg

Let’s get started

QUARKUS

Elite. Java. Development.

group9

Plan of action!

1. Introduction
2. About Quarkus and native compilation
3. How does quarkus help ?
4. Demo
5. What did we do in our project @ASML
6. Conclusions
7.

Introduction

- Ronald Spierenburg
  - Writes software
    - Likes investigating weird problems (although he gets frustrated with them)
    - Is in software consultancy since 2005
  - Born far away from North Brabant
  - Age : 42

- Currently working @ASML on the Integrated Productivity Analytics project.

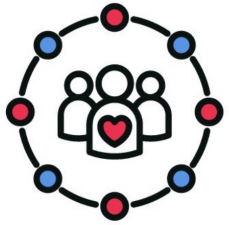
Quarkus and native compilation

- Quarkus

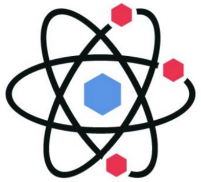
Developer Joy

Kubernetes-native

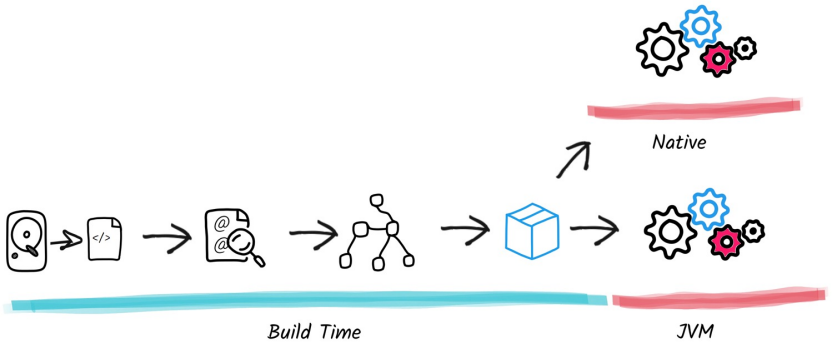
Best of Breed Libraries and Standards



Imperative and reactive code



- Native is a first class resident in Quarkus



Pros and cons

Pros

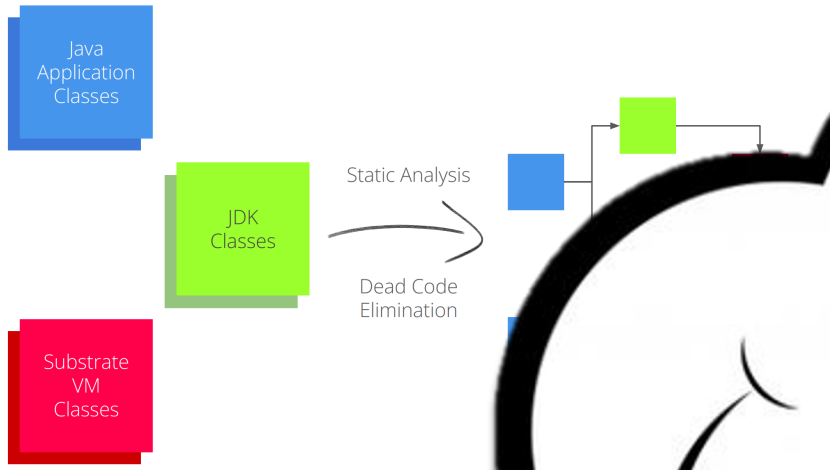
- Faster startup
- Peak performance at beginning
- Small standalone binary
- Memory footprint small (RSS)

Cons

- Slower development cycle
- Lower peak performance
- Security patches require recompilation
- Not portable
- Lacks behind in terms of tooling support

- Or, for ASML : decompilation is way harder

How does it work



GraalVM™

Quarkus can help!

Not supported

- Dynamic classloading
- InvokeDynamic & metho
- Finalizer
- Security manager
- JVMTI, JMX, Native VM interfaces

- l)
- italizers (eager)
- Lamda's, Threads
- References

But Ronald, how does Quarkus help us ?

- Quarkus ecosystem is designed from the core to support native compilation
- All quarkus extensions support have native compilation support
- Property driven customisation (application.properties) or json config file in META-INF/native-image/

```
quarkus.native.resources.includes=foo/**,bar/**/*.txt
quarkus.native.resources.excludes=foo/private/**
```

```
{
  "resources": [
    {
      "pattern": ".*\\.xml$"
    },
    {
      "pattern": ".*\\.json$"
    }
  ]
}
```

But Ronald, how does Quarkus help us ? (2)

- Register for reflection. This is for every DTO!

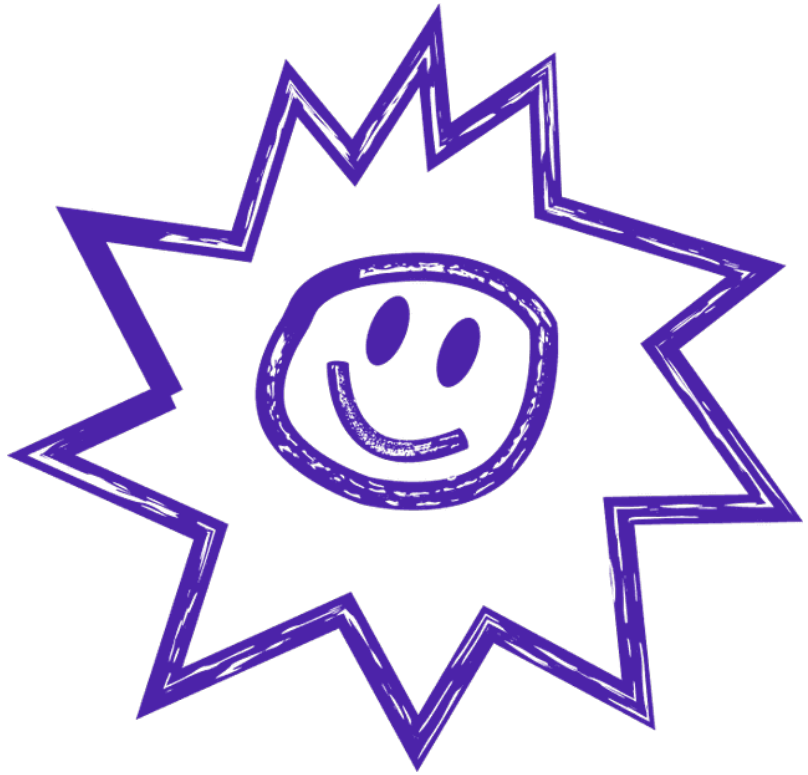
```
@RegisterForReflection
public class MyClass {
}
```

- Delay class initialisation

```
quarkus.native.additional-build-args=--initialize-at-run-
time=com.example.SomeClass\\,org.acme.SomeOtherClass
```

Demo

Project with native compilation



Elite. Java. Development.

group9

What did we do in our project @ASML

- Let's start, this seems easy
- Yay, it works
- Hmm, this does not
- A long time staring at the screen
- Questioning life
- Lot of googling
- A lot of swearing
- Debugging native binaries
- More swearing & sleepless nights
- Work around native compilation
- Try to forget everything that happened

Apache
FOP

What did we do in our project @ASML

1) A @RegisterForReflection quarkus extension
   - Takes all the specified classes and libraries
   - And registers them for reflection in their entirety
   - Because developers can't be trusted and DTO's can reliably be identified
2) An embedded jvm for pdf generation
   1) We were stuck on apache FOP for legacy reasons

```
Apache™ FOP (Formatting Objects Processor) is a print
formatter driven by XSL formatting objects (XSL-FO) and an
output independent formatter. It renders to PDF, PS, PCL,
AFP, XML, Print, AWT and PNG, RTF and TXT.
```

   2) It has no support for GraalVM
   3) Heavily depends on reflection and java.awt
   4) So we ignored it and moved PDF's to a separate embedded microservice

Conclusions

Creating native binaries is easy, as long as:

- You're aware of the quarkus conventions
- You use thorough code inspections, or use brute force @RegisterForReflection
- You don't mind longer build times
- You use @QuarkusIntegrationTest for native integration tests
- And stick to the ecosystem
  - Or at least make GraalVM support a search criterium for dependencies

Any questions?

Elite. Java. Development.

group9