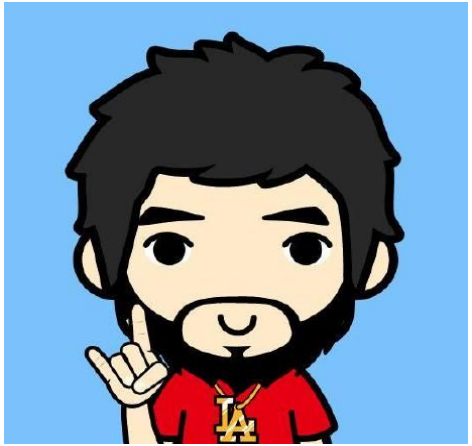
Aplicações Nativas com Java

William Siqueira @ github.com/jesuino



- Banco de Dados @ FATEC São José dos Campos 2010
- Software Engineer @ Red Hat
- Colaborador do JUG Vale jugvale.com
- Colaborador do SJCDigital github.com/sjcdigital
- Escreve em alguns blogs
- Palestrante JavaOne, The Developers Conference, FISL e outros
- Opensource github.com/jesuino

- Review how Java Compilation process works
- What is native compilation
- Why going to native
- Using GraalVM to generate native bits
- GraalVM limitations
- Tools to Port existing Java applications to native
  - Gluon and native JavaFX
  - Quarkus and native Microservices applications
  - Native Business Automation applications with Kogito
- Analysis of a native application

How Java Compilation Process Works

Java Code is compiled to Bytecode and later Bytecode is interpreted by the JVM

```java
public class HelloTDC {

        public static void main(String args[]) {
                System.out.println("Hello TDC!");
        }
}
```

HelloTDC.java

javac HelloTDC.java

```
[wsiqueir@wsiqueir projects]$ javap -c HelloTDC.class
Compiled from "HelloTDC.java"
public class HelloTDC {
  public HelloTDC();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3                  // String Hello TDC!
       5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```

HelloTDC.class

java HelloTDC

Java Bytecode crash course
https://www.youtube.com/watch?v=e2zmmkc5xl0

Any operating system with a supported JVM can run HelloTDC.class

What is native Compilation

Java Code is compiled to Bytecode and Bytecode is compiled to the targeted device binary format

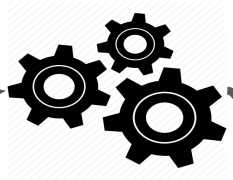
HelloTDC.java

```
1 public class HelloTDC {
2
3         public static void main(String args[]) {
4                 System.out.println("Hello TDC!");
5         }
6 }
```

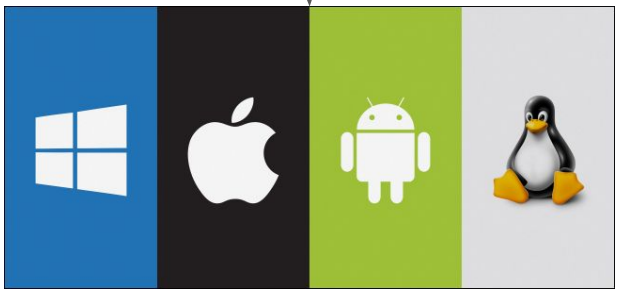
javac HelloTDC.java

```
[wsiqueir@wsiqueir projects]$ javap -c HelloTDC.class
Compiled from "HelloTDC.java"
public class HelloTDC {
  public HelloTDC();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3                  // String Hello TDC!
       5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```



tool to compile to native



a specific binary file or package is generated for the target platform

Why create native Java apps

- Improved performance
- Less requirements on target platform
- Usually smaller applications
- Can run natively on cloud platforms such as **Kubernetes**
  - **Java was losing space in the cloud and serverless architectures due performance reasons. With Quarkus, which we will talk later, we are back in the game!**

GraalVM

High-performance polyglot VM

Can Compile Java applications directly to binary using the Substract VM. But is much more than this!

Compiling Java to Native with GraalVM

```
^C[wsiqueir@wsiqueir projects]$ javac HelloTDC.java
[wsiqueir@wsiqueir projects]$ /opt/graalvm/graalvm-ce-19.1.1/bin/native-image HelloTDC
Build on Server(pid: 31423, port: 33817)*
[hellotdc:31423]    classlist:   6,001.78 ms
[hellotdc:31423]        (cap):   1,214.52 ms
[hellotdc:31423]        setup:   3,170.42 ms
[hellotdc:31423]   (typeflow):   4,853.23 ms
[hellotdc:31423]    (objects):   2,610.91 ms
[hellotdc:31423]   (features):     335.45 ms
[hellotdc:31423]     analysis:   7,905.04 ms
[hellotdc:31423]     (clinit):     160.57 ms
[hellotdc:31423]     universe:     488.02 ms
[hellotdc:31423]      (parse):     923.13 ms
[hellotdc:31423]     (inline):   1,350.50 ms
[hellotdc:31423]    (compile):   8,487.08 ms
[hellotdc:31423]      compile:  11,184.53 ms
[hellotdc:31423]        image:     376.84 ms
[hellotdc:31423]        write:      91.16 ms
[hellotdc:31423]      [total]:  29,425.18 ms
[wsiqueir@wsiqueir projects]$ ./hellotdc
Hello TDC!
```

GraalVM limitations

Dynamic Class Loading / Unloading	Not supported
Reflection	Supported (Requires Configuration)
Dynamic Proxy	Supported (Requires Configuration)
Java Native Interface (JNI)	Mostly supported
Unsafe Memory Access	Mostly supported
Class Initializers	Supported
InvokeDynamic Bytecode and Method Handles	Not supported
Lambda Expressions	Supported
Synchronized, wait, and notify	Supported
Finalizers	Not supported
References	Mostly supported
Threads	Supported
Identity Hash Code	Supported
Security Manager	Not supported
JVMTI, JMX, other native VM interfaces	Not supported
JCA Security Services	Supported

Can you build real world applications with these limitations?

Creating JavaFX Native Applications

Gluon tooling can help you to generated native JavaFX Applications for Android, iOS, Linux and Windows (using GraalVM)

https://gluonhq.com/a-boost-for-java-on-the-client

Creating JavaFX Native Applications

Using Gluon VM for Java 11 you are able to create native Java application from [1]. To create native packages:

mvn clean client:compile client:link

And then run with:

mvn client:run

[1] https://github.com/gluonhq/client-samples

Quarkus


QUARKUS
Supersonic Subatomic Java
A Kubernetes Native Java stack tailored for GraalVM & OpenJDK HotSpot,
crafted from the best of breed Java libraries and standards

Quarkus native applications

Quarkus is a Java platform to create microservices. With its extensions you can create all kind of native applications:

- Database access
- REST APIs
- Business Applications
- Messaging clients
- Microservices (implements Microprofile)
- ...

To generate Quarkus native applications you will need GraalVM. You focus on your code and Quarkus will make what is required to compile your code to a native image prepared to run on Kubernetes.

Quarkus #Facts

- Container First
  - Ready for Docker and Kubernetes
- 10x lighter
  - The final JAR is optimized during and is faster than usual applications
- 100x faster
  - Can be compiled to a native package, resulting in high performance applications
- Low memory usage and fast Startup
- Live Reload
  - Run Quarkus in development and your changes are automatically reloaded, no need to server stop/start
- Developer Joy

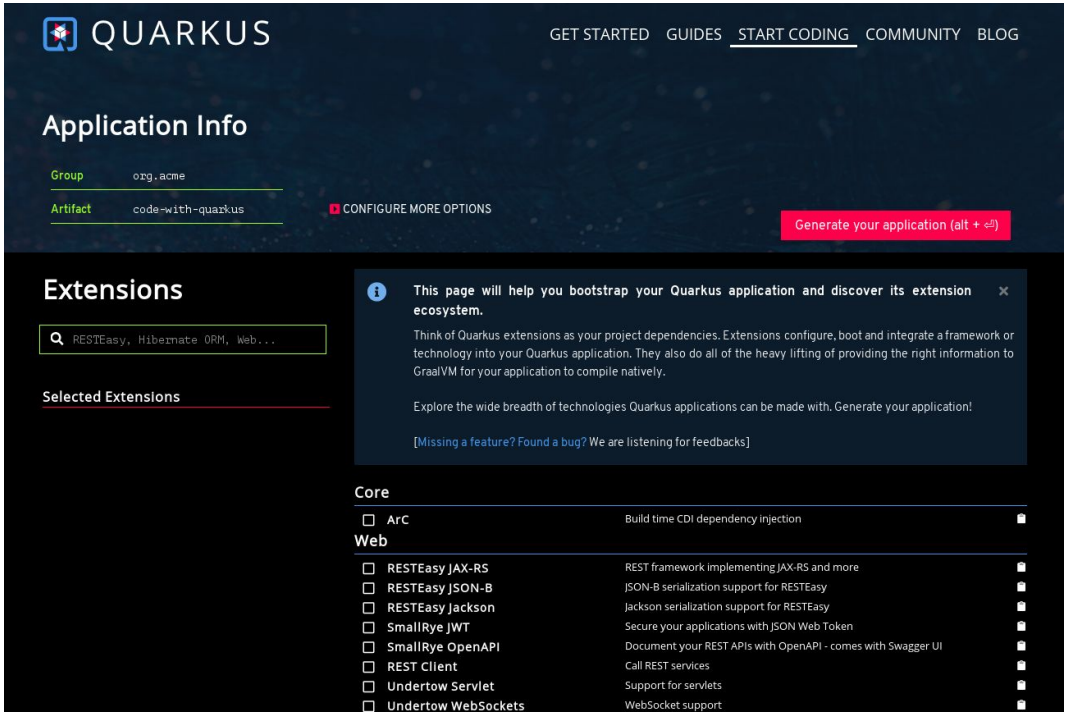
See more in quarkus.io

How fast is Quarkus

it is supersonic, subatomic Java

How can I create Quarkus applications

- Maven and VS Code plugins
- code.quarkus.io



Hello Quarkus application

Business Automation with Quarkus

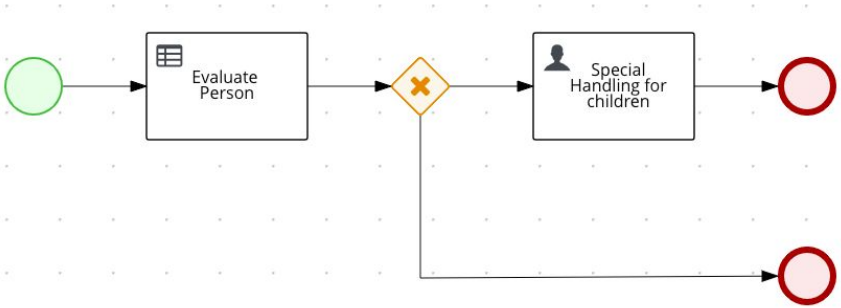
Kogito

You can create native business rules and native business processes using Quarkus with the **Kogito** extension.

jBPM

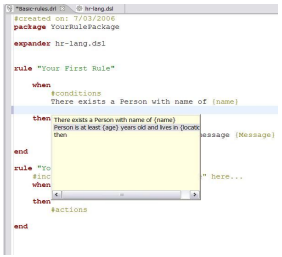
Drools

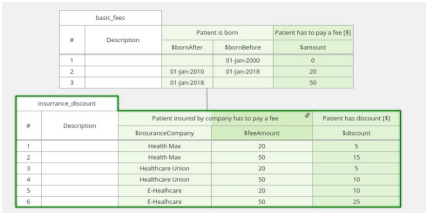
Business Process and Rules?

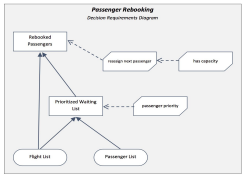


jBPM is a Java platform for creation, execution and management of business processes and cases (dynamic business processes)

Drools is a Java platform to creation, execution and management of business rules using DMN, DRL (domain language), decision tables and more







Kogito and Quarkus brings jBPM and Drools to run natively in the cloud! See kogito.kie.org

Hello Kogito Application

Going native with Quarkus

Quarkus requires GraalVM installation

Get GraalVM 19.2.0.1 for native compilation

Build the project with flag -Pnative

mvn clean install -Pnative

The JUG CFP example

JUG CFP is a real world application: ***mvn clean install -Pnative***

```
[INFO] [io.quarkus.creator.phase.nativeimage.NativeImagePhase] Running Quarkus native-image plugin on Java HotSpot(TM) 64-Bit Server VM
[INFO] [io.quarkus.creator.phase.nativeimage.NativeImagePhase] /opt/graalvm/graalvm-ce-19.1.1/bin/native-image -J-Djava.util.logging.manager=org.jboss.logmanager.LogManager
-J-Dio.netty.leakDetection.level=DISABLED -J-Dvertx.disableDnsResolver=true -J-Dio.netty.noUnsafe=true --initialize-at-build-time= -H:InitialCollectionPolicy=com.oracle.svm.
core.genscavenge.CollectionPolicy$BySpaceAndTime -jar jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar -J-Djava.util.concurrent.ForkJoinPool.common.parallelism=1 -H:FallbackThreshold
=0 -H:+ReportExceptionStackTraces -H:+PrintAnalysisCallTree -H:-AddAllCharsets -H:EnableURLProtocols=http,https --enable-all-security-services -H:-SpawnIsolates -H:+JNI --no
-server -H:-UseServiceLoaderFeature -H:+StackTrace
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]    classlist:  13,183.90 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]        (cap):   1,068.03 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]        setup:   3,607.86 ms
06:29:22,506 INFO  [org.hib.Version] HHH000412: Hibernate Core {5.4.4.Final}
06:29:22,555 INFO  [org.hib.ann.com.Version] HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
06:29:22,602 INFO  [org.hib.dia.Dialect] HHH000400: Using dialect: org.hibernate.dialect.MariaDB103Dialect
06:29:25,409 INFO  [org.jbo.threads] JBoss Threads version 3.0.0.Beta5
06:29:27,001 INFO  [org.xnio] XNIO version 3.7.2.Final
06:29:27,162 INFO  [org.xni.nio] XNIO NIO Implementation Version 3.7.2.Final
06:29:27,303 INFO  [com.arj.ats.arjuna] ARJUNA012170: TransactionStatusManager started on port 35000 and host 127.0.0.1 with service com.arjuna.ats.arjuna.recovery.ActionSta
tusService
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]   (typeflow):  48,753.65 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]    (objects):  26,728.53 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]   (features):   1,394.01 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]     analysis:  81,304.93 ms
Printing call tree to /home/wsiqueir/projects/jug-call4papers/jug-cfp-server/jug-cfp-micro/target/reports/call_tree_jug-cfp-micro-3.0.0-SNAPSHOT-runner_20191012_063102.txt
Printing list of used classes to /home/wsiqueir/projects/jug-call4papers/jug-cfp-server/jug-cfp-micro/target/reports/used_classes_jug-cfp-micro-3.0.0-SNAPSHOT-runner_2019101
2_063109.txt
Printing list of used packages to /home/wsiqueir/projects/jug-call4papers/jug-cfp-server/jug-cfp-micro/target/reports/used_packages_jug-cfp-micro-3.0.0-SNAPSHOT-runner_20191
012_063109.txt
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]     (clinit):   1,273.46 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]     universe:   3,648.16 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]      (parse):   7,520.30 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]     (inline):  13,086.65 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]    (compile):  58,389.44 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]      compile:  82,283.50 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]        image:   6,345.87 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]        write:   1,162.33 ms
[jug-cfp-micro-3.0.0-SNAPSHOT-runner:19530]      [total]: 219,169.85 ms
```

The JUG CFP example

Final package sizes

- **Java: 43M + JVM size**

```
[wsiqueir@wsiqueir jug-cfp-micro]$ du -h target/lib/
39M     target/lib/
[wsiqueir@wsiqueir jug-cfp-micro]$ ll -h target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
-rw-r--r--. 1 wsiqueir wsiqueir 3.2M Oct 12 06:29 target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
```

- **Native: 68M (no JVM required)**

```
[wsiqueir@wsiqueir jug-cfp-micro]$ ll -h target/jug-cfp-micro-3.0.0-SNAPSHOT-runner
-rwxrwxr-x. 1 wsiqueir wsiqueir 68M Oct 12 06:32 target/jug-cfp-micro-3.0.0-SNAPSHOT-runner
```

The JUG CFP example

Startup time

- **Java: 2.780s**

```
[wsiqueir@wsiqueir jug-cfp-micro]$ ll -h target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
-rw-r--r--. 1 wsiqueir wsiqueir 3.2M Oct 12 06:29 target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
[wsiqueir@wsiqueir jug-cfp-micro]$ java -jar target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
2019-10-12 06:45:48,792 INFO  [io.quarkus] (main) Quarkus 0.22.0 started in 2.780s. Listening on: http://[::]:8080
2019-10-12 06:45:48,799 INFO  [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, jdbc-mariadb, maile
r, narayana-jta, resteasy, resteasy-jsonb, security, smallrye-openapi, swagger-ui, vertx]
```

- **Native: 0.509s**

```
[wsiqueir@wsiqueir jug-cfp-micro]$ ./target/jug-cfp-micro-3.0.0-SNAPSHOT-runner
2019-10-12 06:46:25,129 INFO  [io.quarkus] (main) Quarkus 0.22.0 started in 0.509s. Listening on: http://[::]:8080
2019-10-12 06:46:25,129 INFO  [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, jdbc-mariadb, maile
r, narayana-jta, resteasy, resteasy-jsonb, security, smallrye-openapi, swagger-ui, vertx]
```

NOTE: Swagger UI may not be used in production

The JUG CFP example

Memory usage (after first request)

Java: 8579668K

```
[wsiqueir@wsiqueir kie-soup-maven-integration]$ ps -ef | grep cfp
wsiqueir 22226 15848  3 06:58 pts/1    00:00:06 java -jar ./target/jug-cfp-micro-3.0.0-SNAPSHOT-runner.jar
wsiqueir 22612 25152  0 07:02 pts/6    00:00:00 grep --color=auto cfp
[wsiqueir@wsiqueir kie-soup-maven-integration]$ sudo pmap 22226 | grep total
 total          8379976K
```

Native: 1083668K

```
[wsiqueir@wsiqueir kie-soup-maven-integration]$ ps -ef | grep cfp
wsiqueir 22782 15848  0 07:04 pts/1    00:00:00 ./target/jug-cfp-micro-3.0.0-SNAPSHOT-runner
wsiqueir 22890 25152  0 07:05 pts/6    00:00:00 grep --color=auto cfp
[wsiqueir@wsiqueir kie-soup-maven-integration]$ sudo pmap 22782 | grep total
 total          1083668K
```

Conclusion

- Native applications bring better performances for Java cloud applications
- It is not a silver bullet
- The compilation time needs to be considered
- Quarkus makes easy to use real world and complex libraries in native mode

Thanks!