GraalVM

Lviv Java Club

Key Features

High Performance

Apply Graal, an advanced optimizing compiler, that generates faster and leaner code requiring fewer compute resources

See benchmarks

AOT Native Image Compilation

Compile Java applications ahead-of-time to native binaries that start up instantly and deliver peak performance with no warmup time

Learn more

Polyglot Programming

Leverage the best features and libraries of popular languages in a single app with no overhead

Try demos

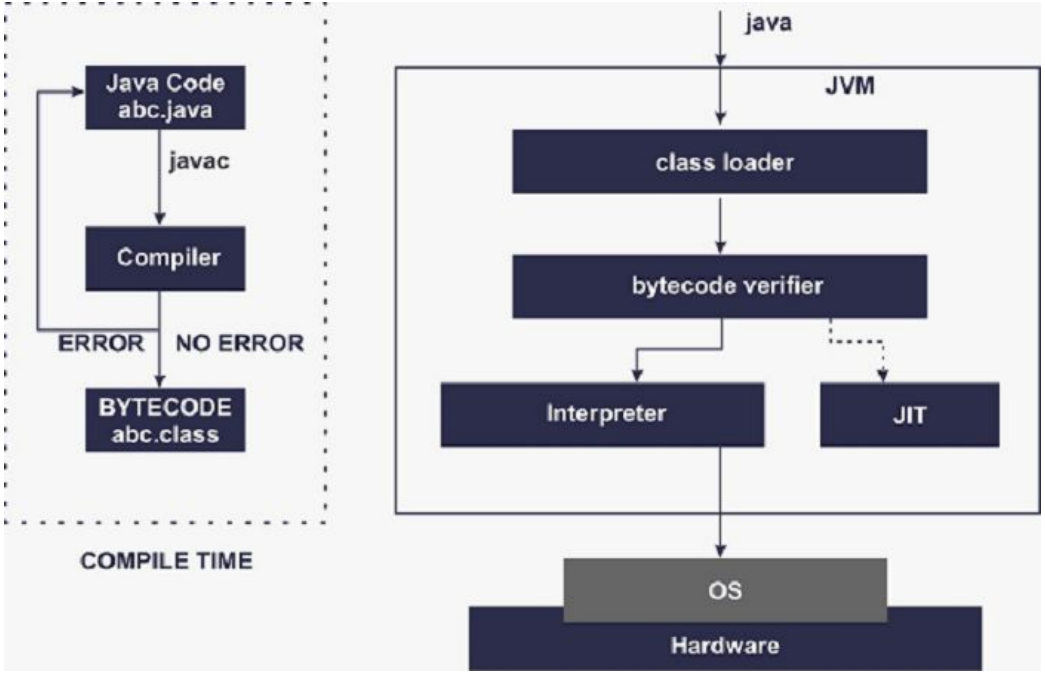
Advanced Tools

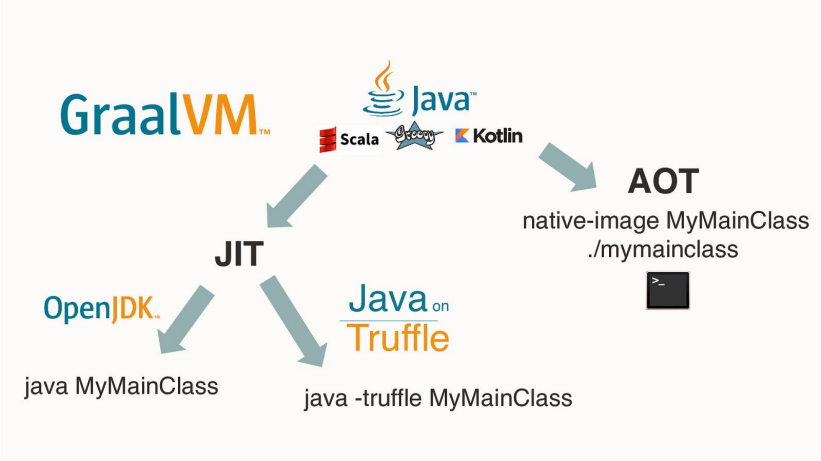
Debug, monitor, profile, and optimize resources consumption in Java and across multiple languages

Read more

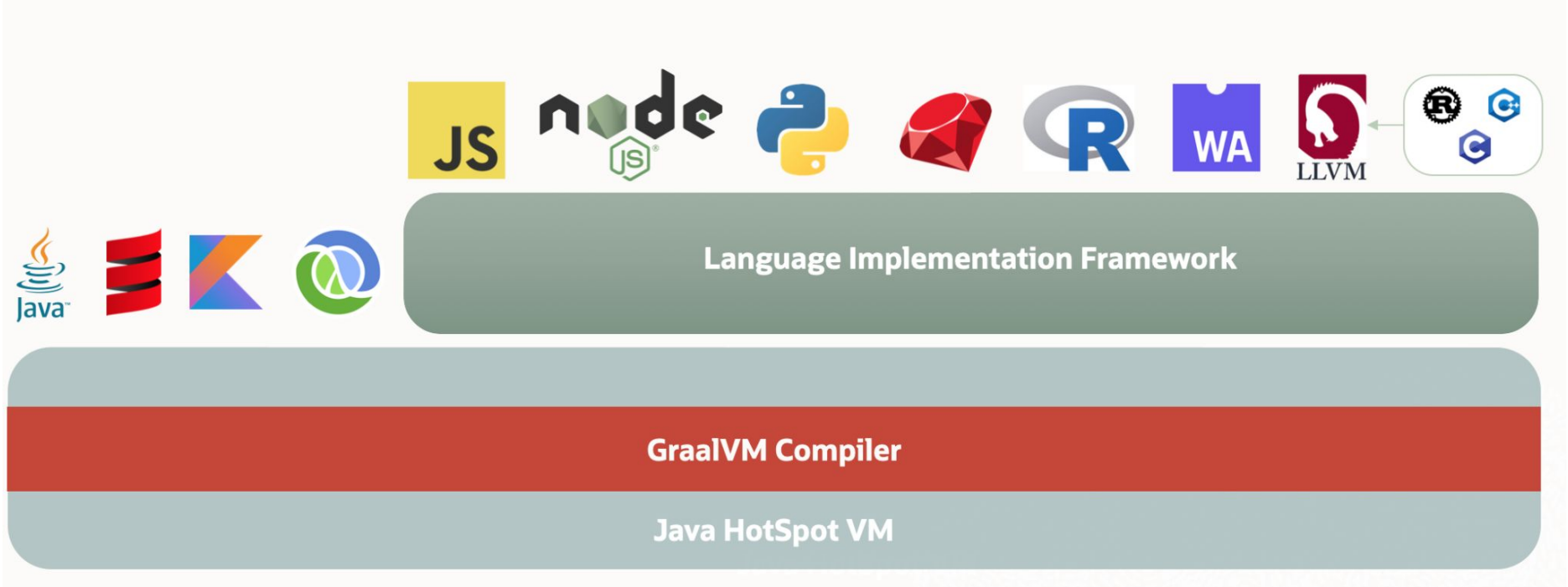
https://www.graalvm.org/

HotSpot vs GraalVM





GraalVM Architecture



Core Components

- Java HotSpot VM
- Graal compiler - the top-tier JIT compiler
- Polyglot API – the APIs for combining programming languages in a shared runtime
- GraalVM Updater - a utility to install additional functionalities

Additional Components

GraalVM core installation can be extended with more languages runtimes and utilities.

Tools/Utilities:

- Native Image – a technology to compile an application ahead-of-time into a native platform executable.
- LLVM toolchain – a set of tools and APIs for compiling native programs to bitcode that can be executed on GraalVM.

Runtimes:

- JavaScript runtime with JavaScript REPL with the JavaScript interpreter
- Node.js – the Node.js 16.14.2 runtime for JavaScript
- LLVM runtime with `lli` tool to directly execute programs from LLVM bitcode
- Java on Truffle – a JVM implementation built upon the Truffle framework to run Java via a Java bytecode interpreter.
- Python – Python 3.8.5 compatible
- Ruby – Ruby 3.0.3 compatible
- R – GNU R 4.0.3 compatible
- GraalWasm – WebAssembly (Wasm)

GraalVM Updater

GraalVM Updater, `gu`, is a command-line tool for installing and managing optional GraalVM language runtimes and utilities. It is available in the core GraalVM installation.
To assist you with the installation, language runtimes and utilities are pre-packaged as JAR files and referenced in the documentation as “components”.

gu list

gu available

gu install js

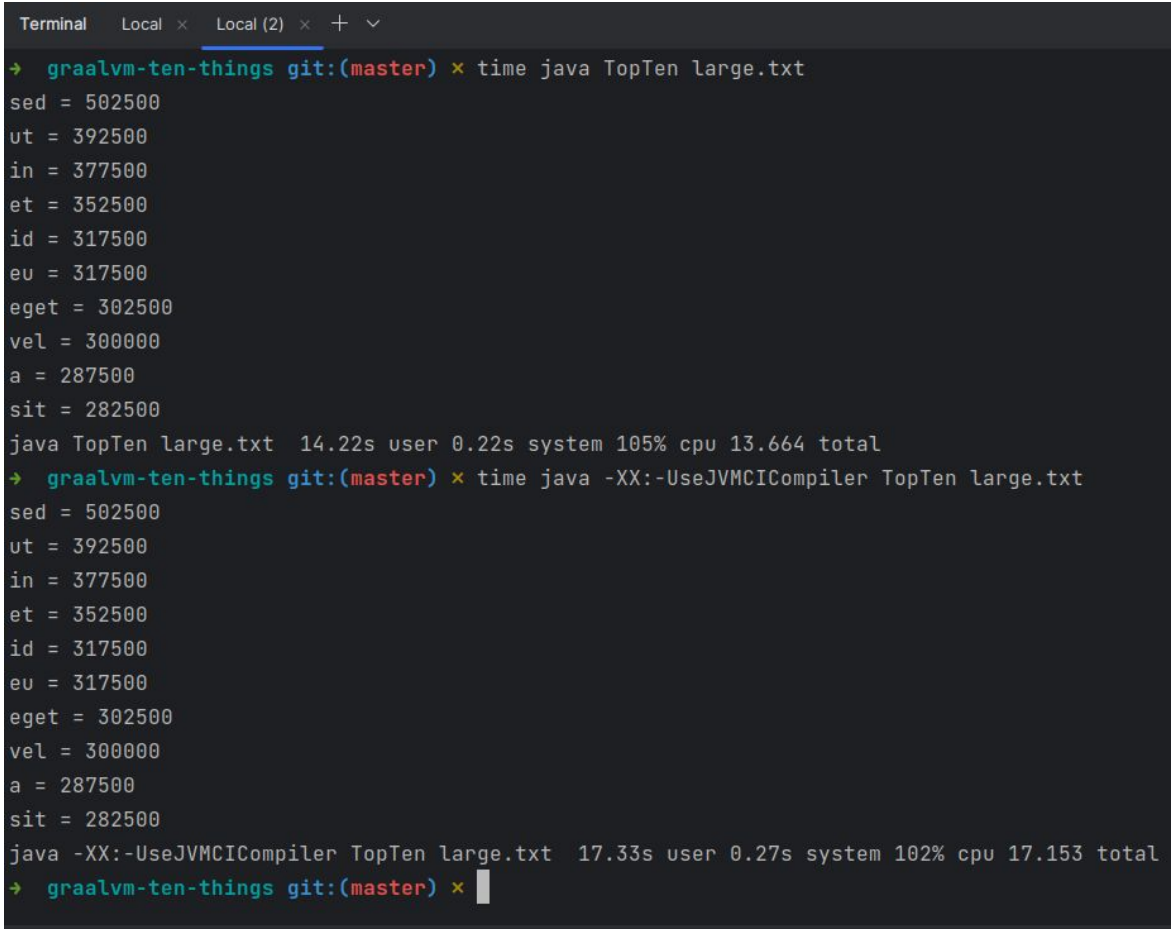
- Check Available Components
- Install Components on GraalVM Community
- Install Components on GraalVM Enterprise
- Install Components Manually
- Install Components from Local Collection
- Uninstall Components
- Upgrade GraalVM
- Rebuild Images
- Replace Components and Files
- Configure Proxies
- Configure Installation
- GraalVM Updater Commands

GraalVM JIT compiler

the large.txt file is 150 MB

program, which gives you the top-ten words in a document

it uses streams and collectors



```
Command being timed: "java TopTen small.txt"
User time (seconds): 0.13
System time (seconds): 0.01
Percent of CPU this job got: 197%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.07
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 74628
```

GraalVM JIT Compiler

```
Command being timed: "java -XX:-UseJVMCICompiler TopTen small.txt"
User time (seconds): 0.18
System time (seconds): 0.00
Percent of CPU this job got: 147%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.12
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 44804
```

Native Image

```
gu install native-image
```

```
javac HelloWorld.java
```

```
native-image HelloWorld
```

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello, Native World!");

    }

}
```

```
➜  graalvm_demo ./helloworld
Hello, Native World!
```

Native Image

```
➜  graalvm-ten-things git:(master) ✗ javac TopTen.java
➜  graalvm-ten-things git:(master) ✗ native-image TopTen
```

```
Command being timed: "./topten small.txt"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 100%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 11500
```

Polyglot Programming

GraalVM allows users to write polyglot applications that seamlessly pass values from one language to another by means of the Truffle language implementation framework

Create the file `polyglot.js`:

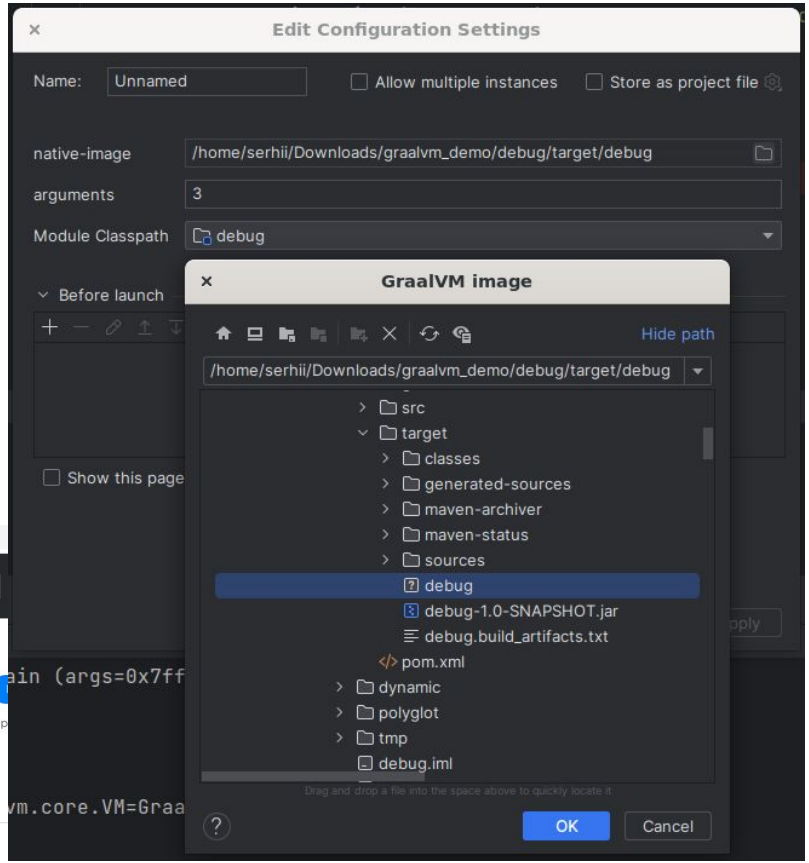
R Ruby Python **Java** LLVM

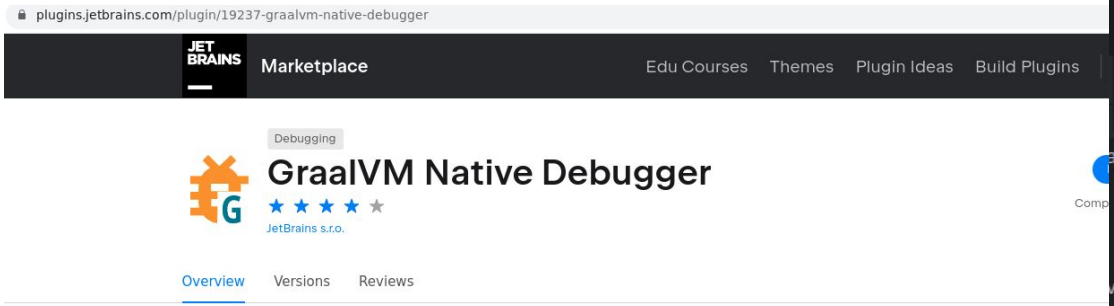
```
var array = new (Java.type("int[]"))(4);
array[2] = 42;
console.log(array[2])
```

https://www.graalvm.org/22.0/reference-manual/polyglot-programming/

Debugging via IntelliJ IDEA

mvn -Pnative -DskipTests package





[GraalVM Native Debugger - IntelliJ IDEA Plugin | Marketplace]

Advanced Tooling with GraalVM

https://www.graalvm.org/advanced-tools/

VS Code Extensions

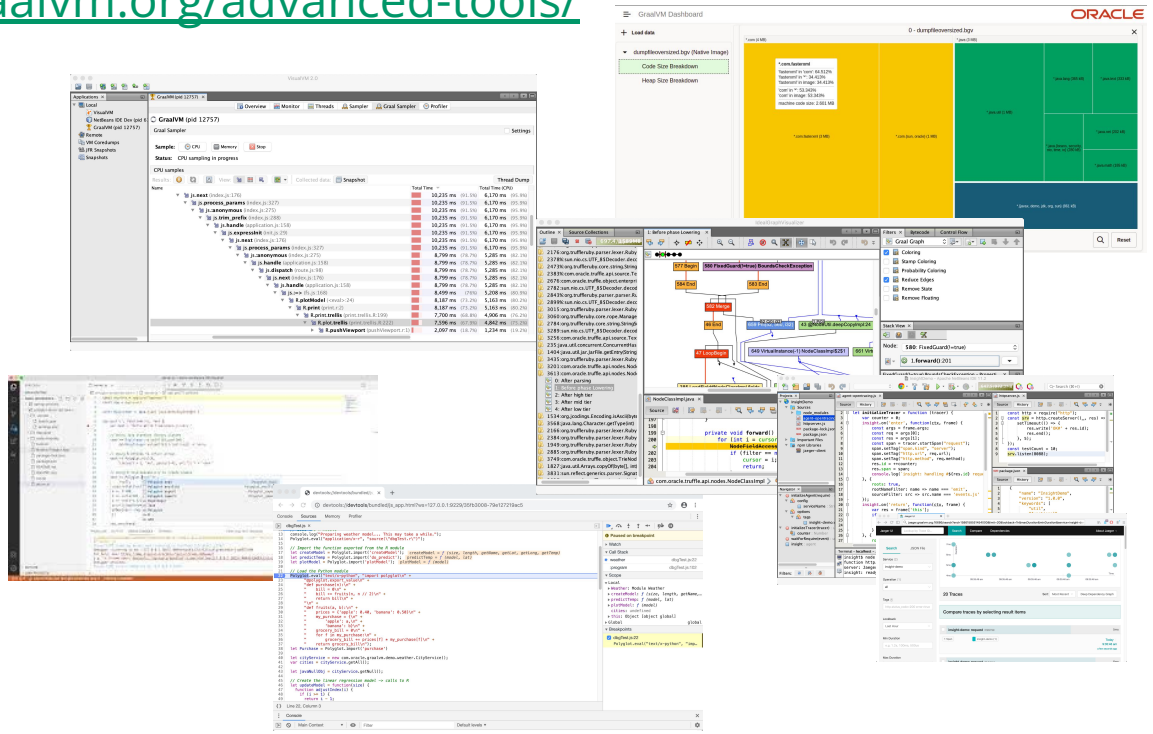
GraalVM Dashboard

Chrome Debugger

VisualVM

GraalVM Insight

Ideal Graph Visualizer



Spring and Native Images

15.1.1. Key Differences with JVM Deployments

The fact that GraalVM Native Images are produced ahead-of-time means that there are some key differences between native and JVM based applications. The main differences are:

- Static analysis of your application is performed at build-time from the `main` entry point.
- Code that cannot be reached when the native image is created will be removed and won't be part of the executable.
- GraalVM is not directly aware of dynamic elements of your code and must be told about reflection, resources, serialization, and dynamic proxies.
- The application classpath is fixed at build time and cannot change.
- There is no lazy class loading, everything shipped in the executables will be loaded in memory on startup.
- There are some limitations around some aspects of Java applications that are not fully supported.

https://docs.spring.io/spring-boot/docs/3.0.0/reference/htmlsingle/#native-image

Spring and Native Images

A closed-world assumption implies the following restrictions:

- The classpath is fixed and fully defined at build time
- The beans defined in your application cannot change at runtime, meaning:
  - The Spring `@Profile` annotation and profile-specific configuration is not supported
  - Properties that change if a bean is created are not supported (for example, `@ConditionalOnProperty` and `.enable` properties).

https://docs.spring.io/spring-boot/docs/3.0.0/reference/htmlsingle/#native-image

Spring and Native Images

```java
@Configuration(proxyBeanMethods = false)
public class MyConfiguration {

    @Bean
    public MyBean myBean() {
        return new MyBean();
    }

}
```

```java
/**
 * Bean definitions for {@link MyConfiguration}.
 */
public class MyConfiguration__BeanDefinitions {

    /**
     * Get the bean definition for 'myConfiguration'.
     */
    public static BeanDefinition getMyConfigurationBeanDefinition() {
        Class<?> beanType = MyConfiguration.class;
        RootBeanDefinition beanDefinition = new RootBeanDefinition(beanType);
        beanDefinition.setInstanceSupplier(MyConfiguration::new);
        return beanDefinition;
    }

    /**
     * Get the bean instance supplier for 'myBean'.
     */
    private static BeanInstanceSupplier<MyBean> getMyBeanInstanceSupplier() {
        return BeanInstanceSupplier.<MyBean>forFactoryMethod(MyConfiguration.class, "myBean").withGenerator(
                (registeredBean) -> registeredBean.getBeanFactory().getBean(MyConfiguration.class).myBean());
    }

    /**
     * Get the bean definition for 'myBean'.
     */
    public static BeanDefinition getMyBeanBeanDefinition() {
        Class<?> beanType = MyBean.class;
        RootBeanDefinition beanDefinition = new RootBeanDefinition(beanType);
        beanDefinition.setInstanceSupplier(getMyBeanInstanceSupplier());
        return beanDefinition;
    }

}
```

Spring Petclinic

```
→  spring-petclinic git:(main) ls -lh target/spring-petclinic-3.0.0-SNAPSHOT.jar | awk '{ print $5; }'
53M
→  spring-petclinic git:(main) ls -lh target/spring-petclinic | awk '{ print $5; }'
167M
→  spring-petclinic git:(main)
```

Useful links

https://www.graalvm.org/

https://docs.oracle.com/en/graalvm/enterprise/22/docs/overview/

https://www.graalvm.org/22.2/reference-manual/native-image/guides/build-spring-boot-app-into-native-executable/

https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html

https://tanzu.vmware.com/developer/guides/graalvm-with-spring/

https://blog.jetbrains.com/idea/2022/06/intellij-idea-2022-2-eap-5/

Дякую ЗСУ!

Слава Україні!