GraalVM Native Image Quick Reference v1

Tested with GraalVM 21.0.0

BUILD A NATIVE IMAGE

Build a native image from a JAR file with all dependencies:

```
native-image [options] -jar myApp.jar
[imagename]
```

Specify classes search path for directories, JARs, ZIPs:

```
-cp jar:com/package/**/myApp.jar
```

Specify the custom main class:

```
-H:Class=MyApp
```

Control classes initialization at build or run time:

```
--initialize-at-build/run-time=
  com.example.MyClass,org.package
```

Include resources matching Java RegEx:

```
-H:IncludeResource=/com/package/**/file.xml
```

Enable HTTPS support:

```
--enable-https
```

Install exit handlers:

```
--install-exit-handlers
```

Include all security service classes:

```
--enable-all-security-services
```

Add all charsets support:

```
-H:+AddAllCharsets
```

Include all timezones pre-initialized:

```
-H:+IncludeAllTimeZones
```

Build a statically linked image with libc implementation:

```
--static --libc=glibc|musl
```

Build a statically linked image with libc dynamically linked (distroless):

```
-H:+StaticExecutableWithDynamicLibC
```

Enable polyglot support:

```
--language:java|js|python|ruby|llvm|wasm
```

Attach a debugger to the build process:

```
--debug-attach=[port]
```

BUILD AN OPTIMIZED NATIVE IMAGE

Run GraalPy applications like any other Python application:

```
graalpy [options] [-c cmd | filename]
```

Use profile-guided optimizations:

```
native-image --pgo-instrument MyApp
Run the image to record the profile: ./myapp
native-image --pgo default.iprof MyApp
```

Select GraalVM's garbage collector implementation:

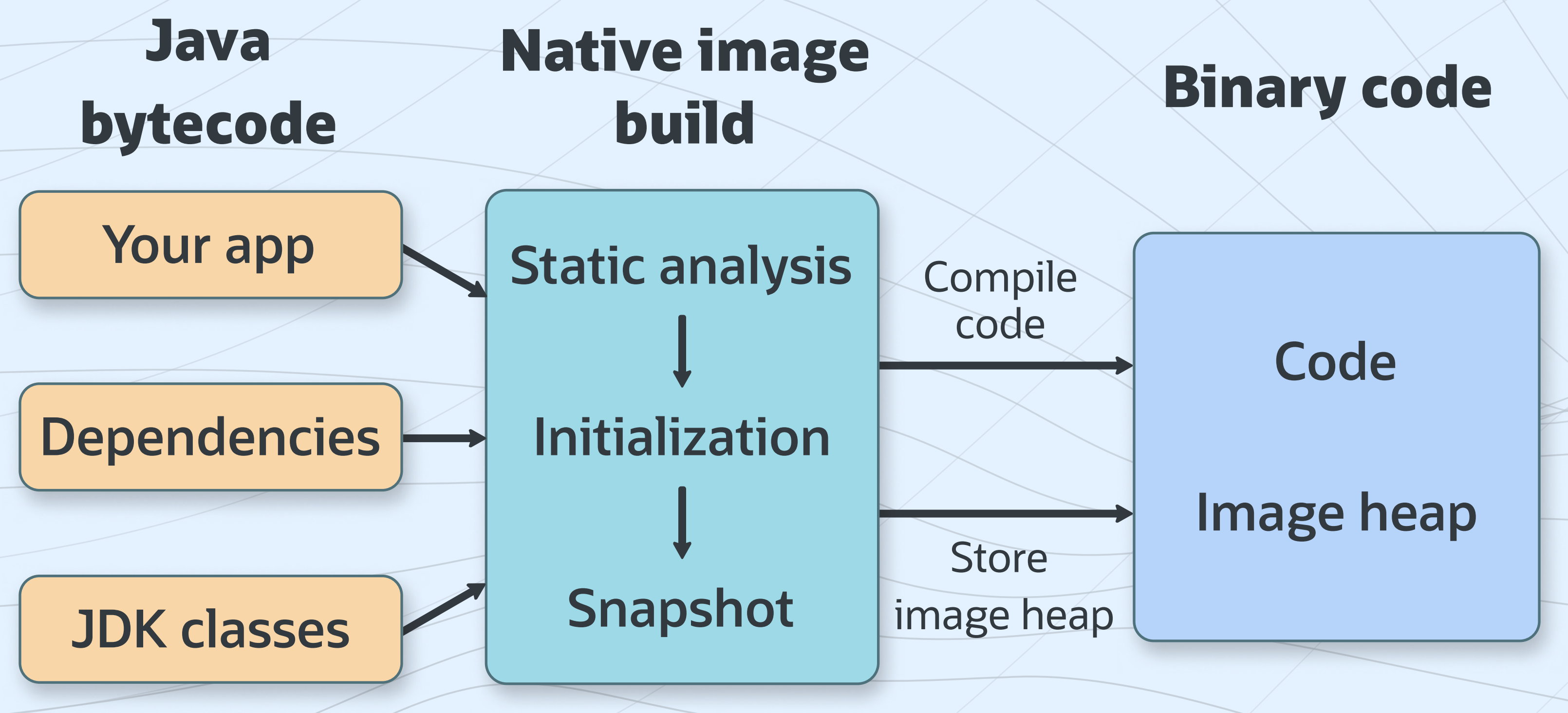
```
--gc=G1
```

BUILD A SHARED LIBRARY

```
native-image -jar jarfile [libraryname]
  --shared
```

Annotate the entry point method(s) with the `@CEntryPoint` annotation. Entry point methods must be static, have non-object parameters and return types.

```
@CEntryPoint
static int add(IsolateThread thread,int a, int b) {
    return a + b;
}
```



More info at: graalvm.org

CONFIGURE A NATIVE IMAGE

Static analysis requires configuration for some language features: accessing resources, serialization, reflection, JNI, etc.

Run a Java process with tracing agent to generate the configuration:

```
java -agentlib:native-image-agent=
  config-output-dir=/path/to/config-dir/
    -jar MyApp.jar
```

Specify the configuration to use for building a native image:

```
-H:ConfigurationFileDirectories=/path/to/
  config-dir/
```

Configuration files in META-INF/native-image on the classpath are included automatically.

Configure memory at run time:

```
./imagename -Xmx<m> -Xmn<m>
```

Configure default heap settings at build time:

```
-R:MaxHeapSize=<m> -R:MaxNewSize=<m>
```

DEBUG A NATIVE IMAGE

Build a native image with debug information:

```
-g
```

Print garbage collection logs:

```
./imagename -XX:+PrintGC -XX:+VerboseGC
```

Trace the initialization path for a certain class:

```
-H:+TraceClassInitialization=
  package.class.Name
```

Print classes intialized detected by the static analysis:

```
-H:+PrintClassInitialization
```

Gather the diagnostic data for GraalVM Dashboard:

```
-H:+DashboardAll   -H:DashboardDump=<path>
```