# GraalVM™ Native Image Quick Reference – GraalVM for JDK 21

## BUILD

Use `native-image` in the same way you would use `java`

to execute a class:

```
native-image [options] <mainclass> [args...]
```

to execute a JAR file:

```
native-image [options] -jar <jarfile> [args...]
```

to execute the main class in a module:

```
native-image [options] -m <module>[/<mainclass>]
[args...]
```

## BUILD OPTIONS

Enable the quick build mode for development:

```
-Ob
```

Specify a name for the resulting binary:

```
-o <myapp>
```

Build a shared library:

```
--shared
```

Build a statically linked binary with the `libc` implementation [2]:

```
--static --libc=musl
```

Pass a comma-separated list of URL protocols:

```
--enable-url-protocols=<http | https |
file | resource | JAR>
```

Specify a list of packages and classes to be linked at build time:

```
--link-at-build-time=<comma-separated
list of packages and classes>
```

Specify a garbage collector:

```
--gc=<serial | G1[1][2] | epsilon>
```

Install exit handlers:

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

Control class initialization at build or run time:

```
--initialize-at-build-time=<comma-
separated list of packages and classes>
```

```
--initialize-at-run-time=<comma-
separated list of packages and classes>
```

Embed a Software Bill of Materials (SBOM)[1]:

```
--enable-sbom
```

Enable Profile-Guided Optimization (PGO) to improve performance and throughput[1]:

1. Build an instrumented native executable:

```
native-image --pgo-instrument MyApp
```

2. Run the executable to record profiles:

```
./myapp
```

3. Build an optimized native executable:

```
native-image --pgo=default.iprof MyApp
```

List all options for `native-image`:

```
--help
```
or
```
--help-extra
```

## INTEGRATIONS

Use the **Gradle** plugin for GraalVM Native Image:

https://graalvm.github.io/native-build-tools/latest/gradle-plugin

Use the **Maven** plugin for GraalVM Native Image:

https://graalvm.github.io/native-build-tools/latest/maven-plugin

Use the **GitHub Action** for GraalVM:

https://github.com/marketplace/actions/github-action-for-graalvm

## CONFIGURE

There are three ways to configure Native Image for external Java libraries:

1. Configuration is provided by a framework such as Micronaut, Helidon, Spring, and Quarkus
2. Configuration is automatically pulled in by the Maven/Gradle plugin from GraalVM Reachability Metadata Repository
3. Configuration is provided manually. The Tracing Agent can help in this process:

a. Run a Java application with the agent to generate the configuration:

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

b. Build a native executable with the configuration:

```
native-image -jar MyApp.jar
```

**Configuration files in META-INF/native-image on the class path or module path are included automatically.**

## MONITOR

Tune the garbage collector:

```
./myapp -Xmx<m> -Xmn<m> …
```

Gather garbage collection logs:

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

Enable application monitoring features (defaults to `all`):

```
--enable-monitoring=heapdump, jfr, jvmstat,
jmxserver, jmxclient, nmt, threaddump
```

Enable JFR support and record events:

```
./myapp -XX:+FlightRecorder
-XX:StartFlightRecording="filename=recording.jfr"
```

Dump the initial heap of a native executable:

```
./myapp -XX:+DumpHeapAndExit
```

## DEBUG

Build a native executable with debug information and with compiler optimizations disabled[2]:

```
-g -O0
```

**Debug a native executable with GDB or any IDE that integrates GDB.**

---

[1] Available with Oracle GraalVM

[2] Linux only