

## BUILD

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 binary with the `libc` implementation <sup>[2]</sup>:

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

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

```
-H:+StaticExecutableWithDynamicLibC
```

Attach a debugger to the build process:

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

## OPTIMIZE

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

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
```

Select GraalVM's garbage collector implementation<sup>[1][2]</sup>:

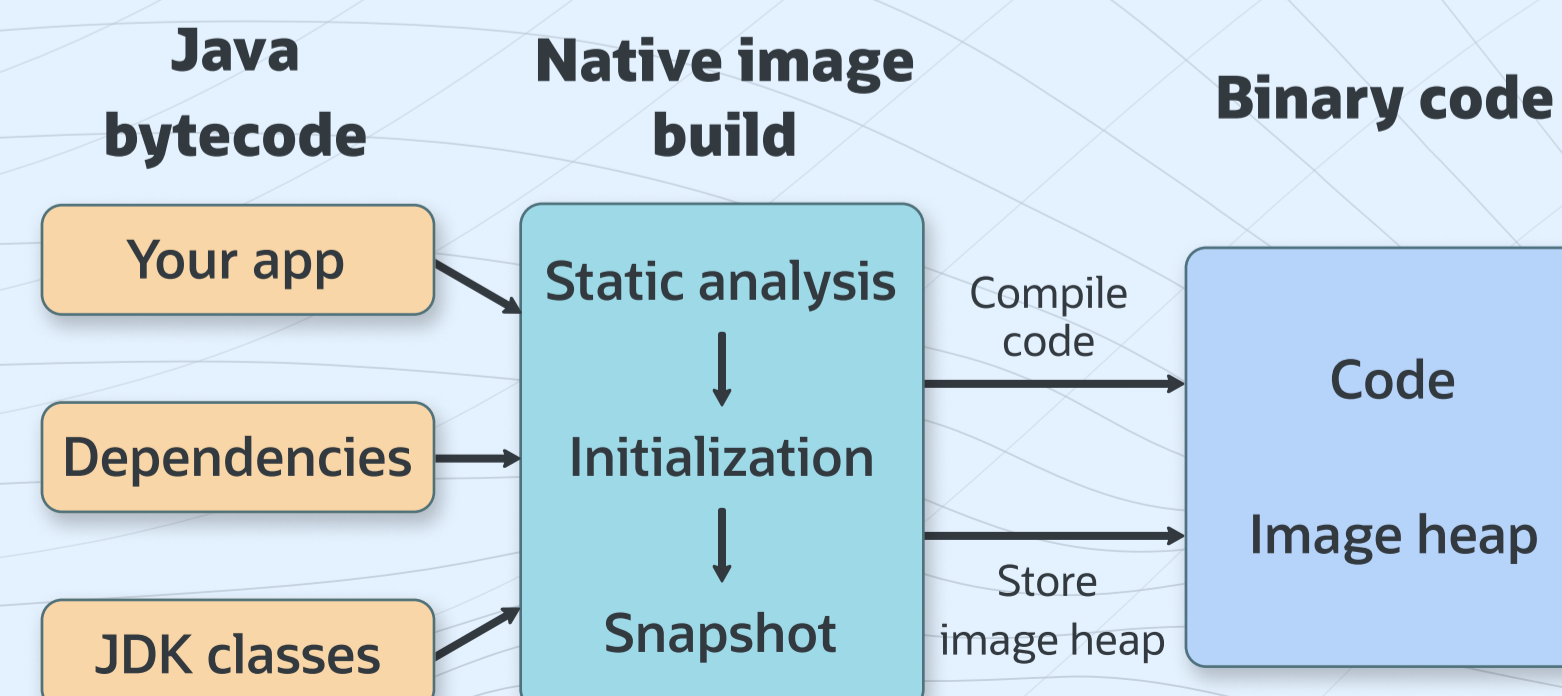
```
--gc=G1
```

## 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](https://graalvm.org)

## CONFIGURE

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

Run a Java process with the 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>
```

## DEBUG

Build a native image with debug information<sup>[2]</sup>:

```
-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 initialized detected by the static analysis:

```
-H:+PrintClassInitialization
```

<sup>[1]</sup> Available with Oracle GraalVM

<sup>[2]</sup> Linux only

