Take Control of Your Build: Spring Boot and Beyond with Mill

Vasilis Nicolaou

Vaslabs LTD

CERN @ 02 December 2025

Outline

1. Introduction
2. Mill in Action
   - Demo 1: Spring Boot with Mill
   - Demo 2: AOT and GraalVM
3. How Mill Works Under the Hood
4. Annotation Processors
5. Android
   - Mill Under Extreme Conditions
6. Closing Remarks
   - Key Takeaways
   - Q&A

Who am I?

- JVM developer (Java, Scala) with experience in distributed systems.
- Contributor to Mill, initially adding Android support and lately around Spring Boot.
- I went through various build tools; Maven, Gradle, Sbt and now... Mill!
- Today: Software Engineering Consultant with own firm and a small but strong team of software engineers!

Demo 1: A simple Spring Boot project

Quick Mill setup commands:

```
url=https://repo1.maven.org/maven2/com/lihaoyi/mill-dist/1.1.0-RC2-40-0576e4/mill-dist-1.1.0-RC2-40-0576e4-mill.sh
curl -L "${url}" -o mill
chmod +x mill
./mill version
```

Basic Spring Boot setup

```scala
package build
import mill.*, mill.javalib.*

object `package` extends MavenModule { outer =>
  override def bomMvnDeps = Seq(
    mvn"org.springframework.boot:spring-boot-dependencies:3.5.7"
  )
  override def mvnDeps = Seq(
    mvn"org.springframework.boot:spring-boot-starter-web"
  )

  object test extends MavenTests, TestModule.Junit5 {
    override def bomMvnDeps = outer.bomMvnDeps()
    override def mvnDeps =
      Seq(mvn"org.springframework.boot:spring-boot-starter-test")
  }
}
```

What's happening under the hood?

```
./mill visualizePlan compile
```

Dev tooling

```
package build
import mill.*, mill.javalib.*, palantirformat.PalantirFormatModule
object `package` extends MavenModule,PalantirFormatModule
```

Or without build code changes

```
./mill mill.javalib.palantirformat.PalantirFormatModule/
```

Some common dev tools

- Linting and formatting: https://mill-build.org/mill/javalib/linting.html#_autoformatting_with_palantir_java_format
- Docker: https://mill-build.org/mill/contrib/docker.html
- Packaging and Publishing: https://mill-build.org/mill/javalib/publishing.html

Demo 2: Spring Boot AOT + GraalVM

- Recently merged support in Mill for:
  - Spring Boot AOT processing.
  - Building native images with GraalVM.

Demo 2: Spring Boot AOT + GraalVM

```
package build
import mill.*, mill.javalib.*, spring.boot.SpringBootModule

object `package` extends SpringBootModule, MavenModule {
  override def springBootPlatformVersion = "3.5.7"
  override def artifactName: T[String] = "spring-boot-native-demo"
  object prod extends SpringBootOptimisedBuildModule, MavenModule
  object native extends MavenModule, NativeSpringBootBuildModule {
    def jvmId = "graalvm-community:23.0.1"
  }
  def mvnDeps =
    Seq(mvn"org.springframework.boot:spring-boot-starter-web")
  object test extends SpringBootTestsModule, MavenTests,
      TestModule.Junit5 {
    def mvnDeps =
      Seq(mvn"org.springframework.boot:spring-boot-starter-test")
  }
}
```

Mill design principles

- Build definitions are Scala code:
  - Uses best world of **OOP** and **FP**.
  - Easily refactorable and testable; each **Task** yields a value that can be inspected, cached, reused.
- Task graph is generated automatically and is inspectable.
  - No need to worry about caching or cleaning up task directories or task files
  - No need to declare dependencies between tasks.
- Modules are composable units:
  - Inheritance and composition to extend or build new functionality.
- Mill gives you the tools to debug issues when creating your own extensions (e.g. slowness - check **mill-chrome-profile.json**)

Demo: Add Task, inspect its directory and value

```scala
trait FileCounterMavenModule extends MavenModule {

  def countAllSourceFiles: T[Int] = Task {
    allSourceFiles().filter(p => os.isFile(p.path)).size
  }

  def allGeneratedSourceFiles: T[Seq[PathRef]] = Task {
    generatedSources().flatMap(p =>
        os.walk(p.path).filter(os.isFile(_))
    ).map(PathRef(_))
  }

  def countGeneratedSourceFiles: T[Int] = Task {
    allGeneratedSourceFiles().size
  }
}
```

Java annotation Processors

- Recently added support for easily adding annotation processors in Mill.
- Annotation processor dependency resolution also takes into account Maven BOMs.
- Useful for Spring Boot configuration metadata generation, Lombok, AutoValue, Dagger, Immutables, etc.
- Example: `https://mill-build.org/mill/javalib/web-examples.html#_micronaut_hello_world_app`

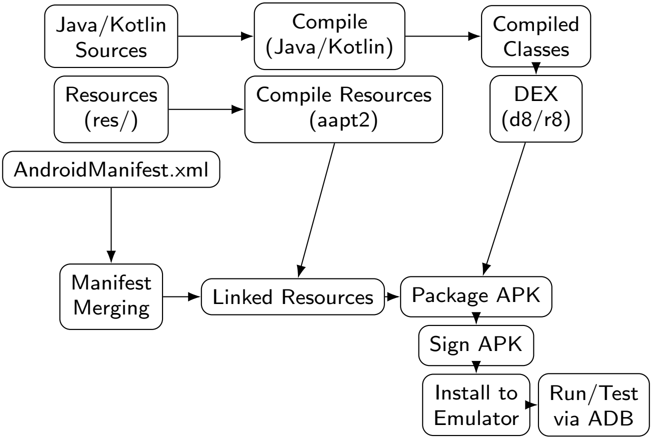
Kotlin: KSP support

- Kotlin Symbol Processing(KSP) support was added as part of the ongoing efforts to add Android support in Mill.
- Can be used for both Kotlin and Android projects.
- Micronaut example: `https://mill-build.org/mill/kotlinlib/web-examples.html#_micronaut_app`
- Dagger example: `https://mill-build.org/mill/kotlinlib/module-config.html#_dependency_injection_with_dagger_and_ksp`

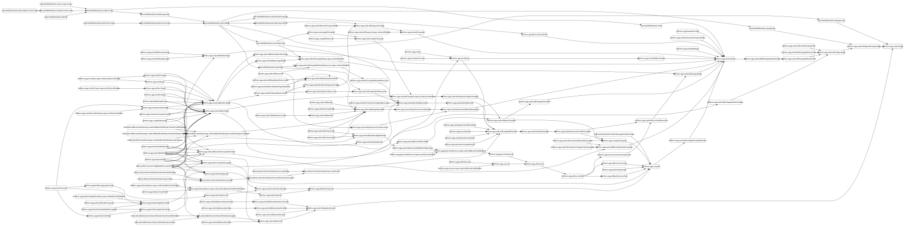
Why Android support in Mill matters

- My first experience with Mill was to add support for Android builds: `https://github.com/com-lihaoyi/mill/pull/4169`
- We've developed the basic feature set for the dev lifecycle and got it merged within 2 weeks.
- About a year since, Mill supports all kinds of complex Android configurations out of the box.
- Android builds are inherently complex.
- Extending Android to support Mill brought functionality for:
  - Dependency resolution with Gradle modules and resolution parameters (e.g. taking into account platform, architecture, build variant).
  - Improved Kotlin tooling with KSP support shown previously (v1 and v2).

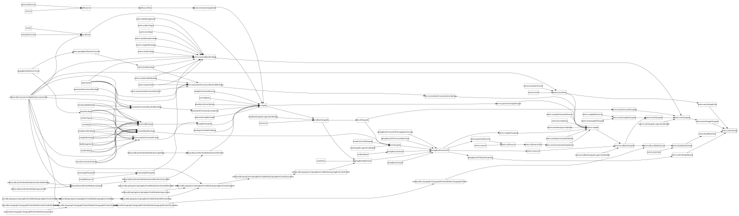
High-level Android pipeline in Mill (conceptual)



Visualizing the task graph for androidApk



Spring Boot native.nativeImage comparison



Key Takeaways

- Mill treats builds as real code, not opaque configuration (but if you wish, you can also use the latter with YAML (new)).
- The task graph is explicit, inspectable, and composable.
- Mill can support modern JVM apps with AOT and native configurations.
- Android support demonstrates that Mill scales to highly complex build pipelines.
- You can take control of your build by easily understanding and extending it.
- Rich examples are documented for a variety of use cases and always validated and tested against each change in Mill.

Summary

Mill is a Swiss knife. But there are efforts to have also the option for non programmatic usage if one wishes to do so!

Questions

- Which version of Spring boot are you using?
- What is your library stack? (Mono/Flux, rxJava, Lombok, Immutables)
- Do you deploy native images? Are you using AOT?
- Any other testing libraries/pre-processors?
- What is the biggest pain on the build tool you are using?
- Plain Java? Kotlin? Mixed?

Thank you!

Thank you!

- Documentation: `https://mill-build.org/`
- Github Repo: `https://github.com/com-lihaoyi/mill`
- Try Mill and join Mill's Discord Channel: