Speed Up Your Maven Build x10...Before You Move to Gradle

Sergei Chernov

Build tool engineering

https://www.linkedin.com/in/schernov/

https://github.com/seregamorph

miro

adyen

Gradle is faster, but...

If you now use Maven, there is a huge space for optimization

Maven is still a pretty good option

If you plan to switch to Gradle, it could make sense to use these advices first

We will talk only about compilation, not tests

Use proper JDK for your CPU arch

```
mvn -v

Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: /opt/homebrew/Cellar/maven/3.9.6/libexec
Java version: 17.0.10, vendor: Amazon.com Inc., runtime: /Users/sergei/Java/amazon-corretto-17.jdk/Contents/Home
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "14.5", arch: "aarch64", family: "mac"
```

If you have ARM-based laptop

```
mvn -v

Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: /opt/homebrew/Cellar/maven/3.9.6/libexec
Java version: 17.0.11, vendor: Oracle Corporation, runtime: /Users/morph/Java/oracle-x64-jdk-17.0.11.jdk
    /Contents/Home
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "14.5", arch: "x86_64", family: "mac"
```

Simple plugin verifying JVM arch vs CPU arch

Add to the root project

```xml
<build>
    <plugins>
        <plugin>
            <groupId>com.github.seregamorph</groupId>
            <artifactId>arch-maven-plugin</artifactId>
            <version>0.1</version>
            <inherited>false</inherited>
            <executions>
                <execution>
                    <id>arch</id>
                    <goals>
                        <goal>arch</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Will fail the build if executed with Rosetta emulation

https://github.com/seregamorph/arch-maven-plugin

Upgrade JDK

And try different vendors

JDK11 is faster than JDK8

JDK17 is faster than JDK11

JDK21 is faster than JDK17

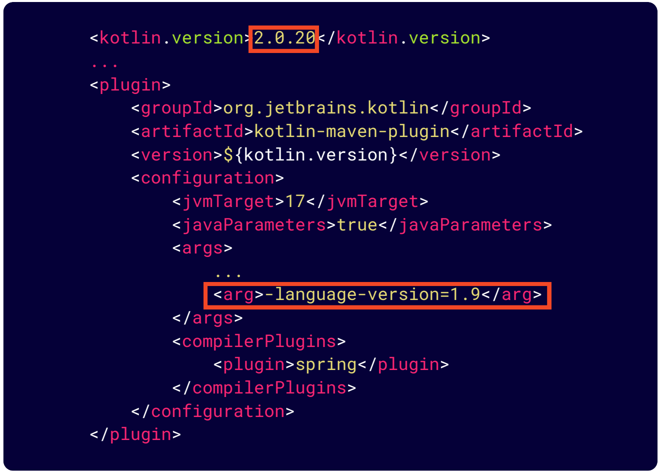
Your results may vary

Kotlin K2 compiler (since Kotlin 2.0)

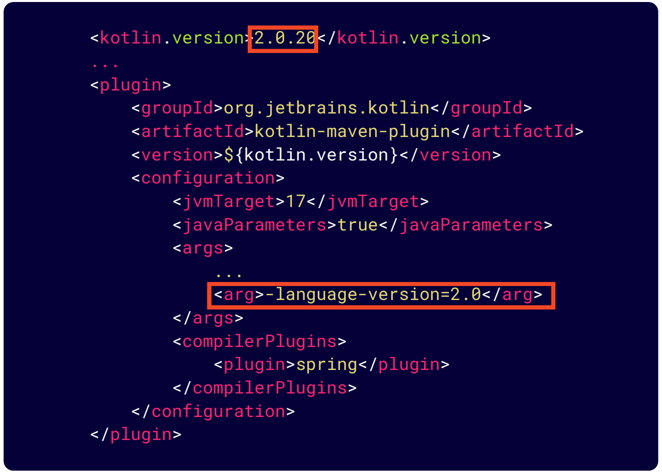
JetBrains rewrote the compiler

Difference is visible even if you mix Java+Kotlin code in the module

First migrate to 1.9.25

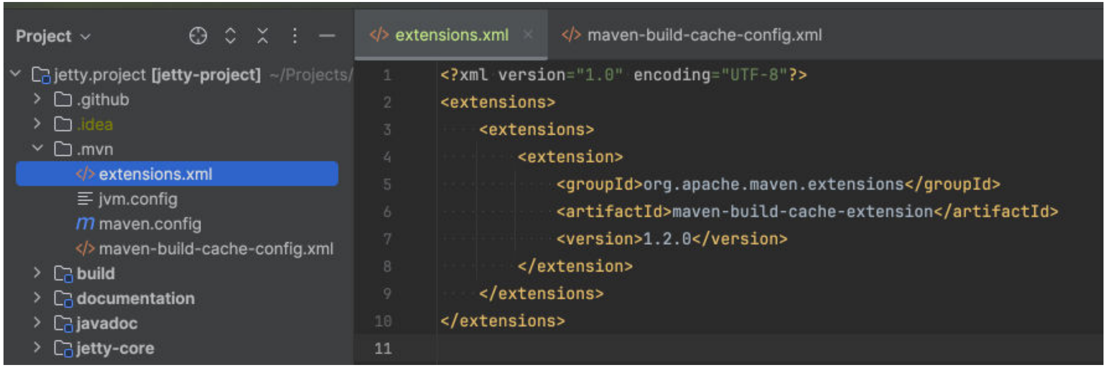


```xml
<kotlin.version>2.0.20</kotlin.version>
...
<plugin>
    <groupId>org.jetbrains.kotlin</groupId>
    <artifactId>kotlin-maven-plugin</artifactId>
    <version>${kotlin.version}</version>
    <configuration>
        <jvmTarget>17</jvmTarget>
        <javaParameters>true</javaParameters>
        <args>
            ...
            <arg>-language-version=1.9</arg>
        </args>
        <compilerPlugins>
            <plugin>spring</plugin>
        </compilerPlugins>
    </configuration>
</plugin>
```



```xml
<kotlin.version>2.0.20</kotlin.version>
...
<plugin>
    <groupId>org.jetbrains.kotlin</groupId>
    <artifactId>kotlin-maven-plugin</artifactId>
    <version>${kotlin.version}</version>
    <configuration>
        <jvmTarget>17</jvmTarget>
        <javaParameters>true</javaParameters>
        <args>
            ...
            <arg>-language-version=2.0</arg>
        </args>
        <compilerPlugins>
            <plugin>spring</plugin>
        </compilerPlugins>
    </configuration>
</plugin>
```

Maven build cache extension by Apache



```
<?xml version="1.0" encoding="UTF-8"?>
<extensions>
    <extensions>
        <extension>
            <groupId>org.apache.maven.extensions</groupId>
            <artifactId>maven-build-cache-extension</artifactId>
            <version>1.2.0</version>
        </extension>
    </extensions>
</extensions>
```

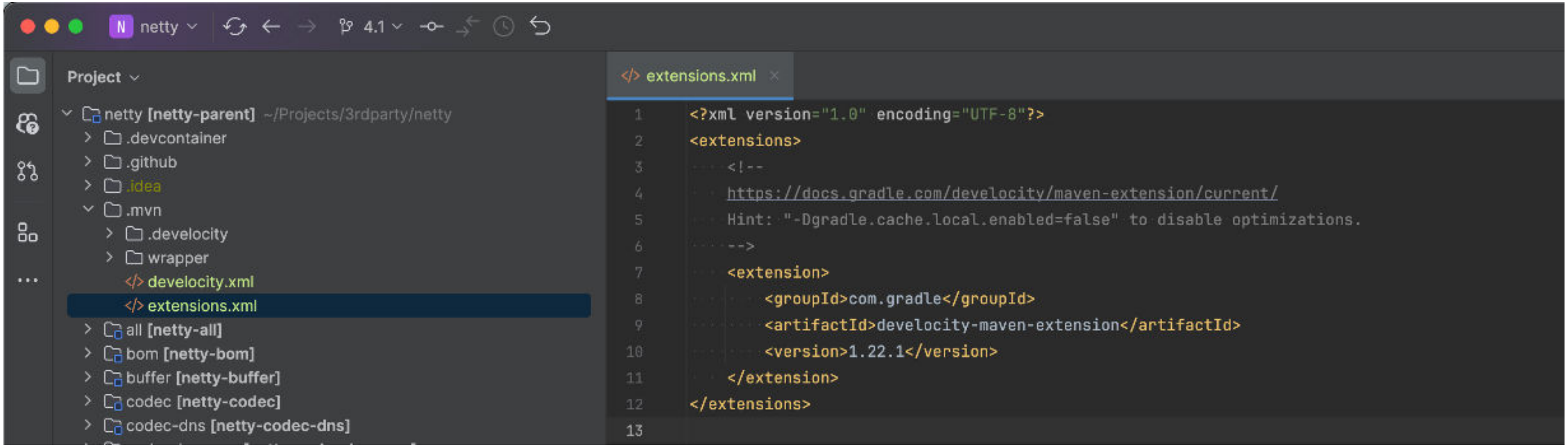
Local and remote caches

Needs detailed configuration, fragile

Does not support test distribution, test selection, etc.

No buid scans

Develocity (former Gradle Enterprise) Extension



```xml
<?xml version="1.0" encoding="UTF-8"?>
<extensions>
    <!--
    https://docs.gradle.com/develocity/maven-extension/current/
    Hint: "-Dgradle.cache.local.enabled=false" to disable optimizations.
    -->
    <extension>
        <groupId>com.gradle</groupId>
        <artifactId>develocity-maven-extension</artifactId>
        <version>1.22.1</version>
    </extension>
</extensions>

```

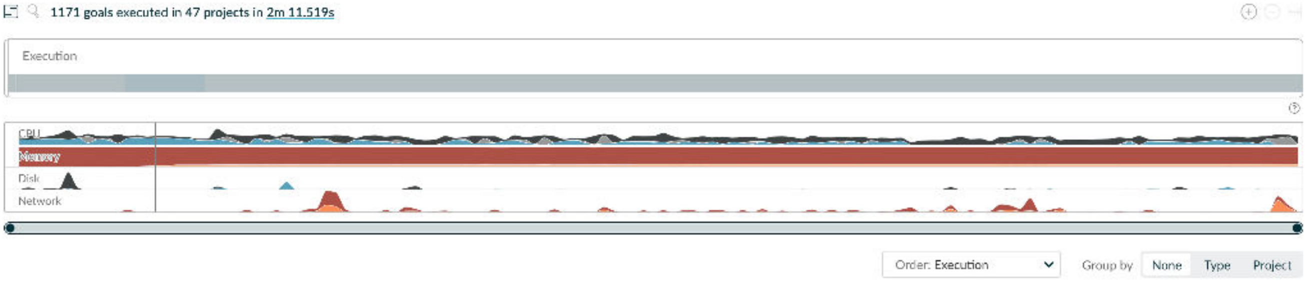
Local and remote build cache

Build scans

Boosts build out-of-the-box

Without cache

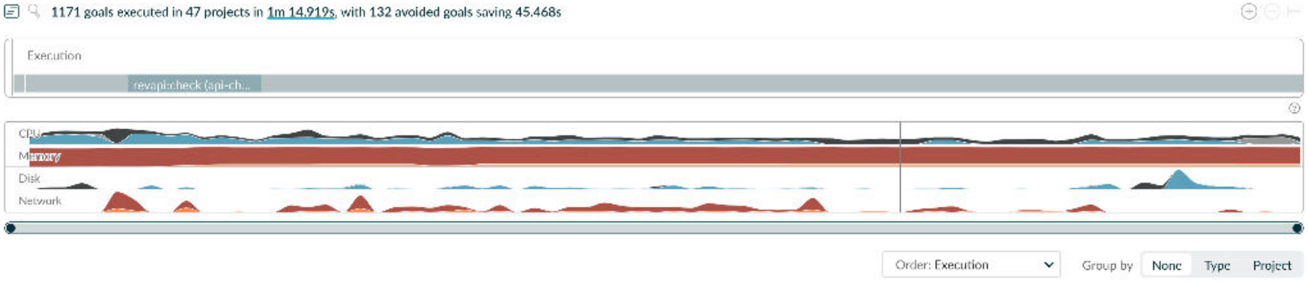
```
mvn clean install -DskipTests=true -Dgradle.cache.local.enabled=false
```



1171 goals executed in 47 projects in 2m 11.519s

VS with the cache

```
mvn clean install -DskipTests=true
```

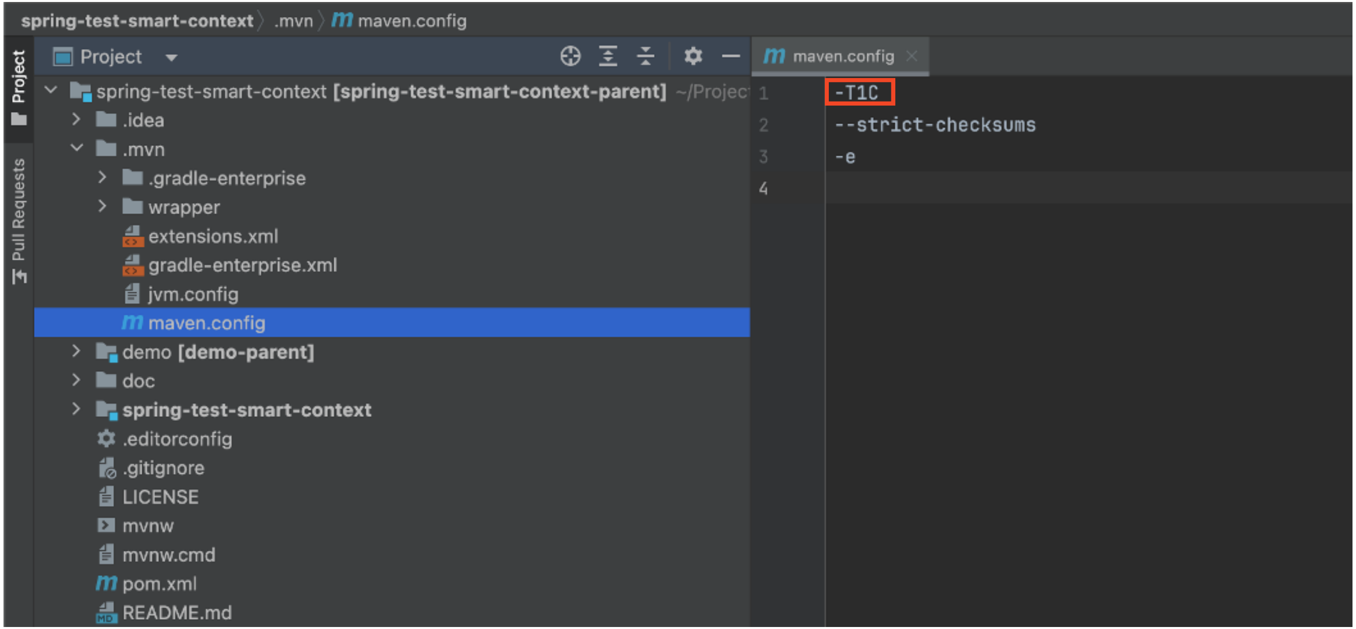


1171 goals executed in 47 projects in 1m 14.919s, with 132 avoided goals saving 45.468s

Maven parallel build

```
mvn clean install -T6
mvn clean install -T1C
```

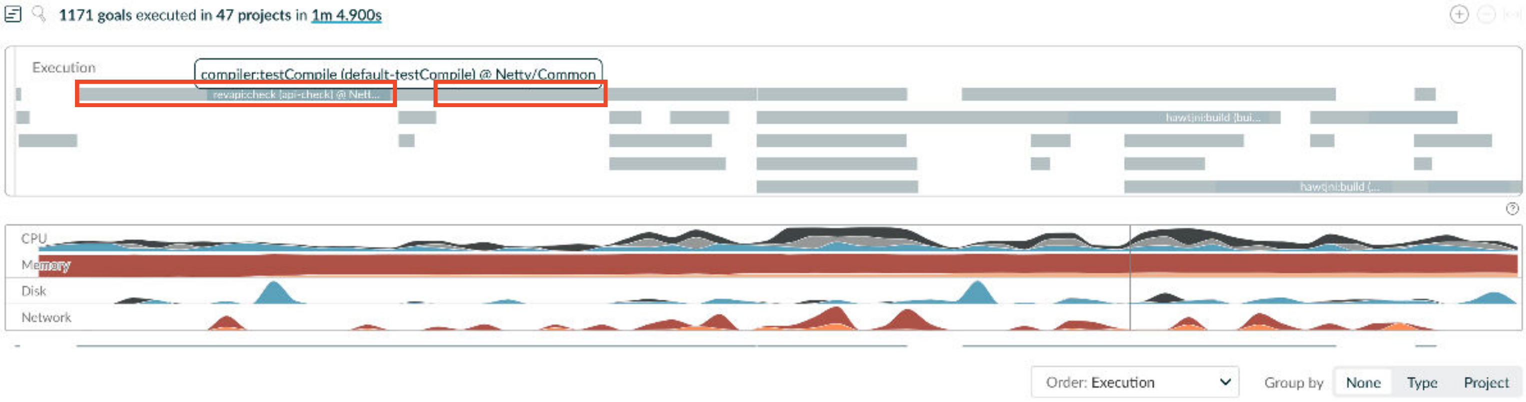
make sure your tests support parallel execution (fixed ports, etc.)



```
-T1C
--strict-checksums
-e
```

Visualize bottlenecks

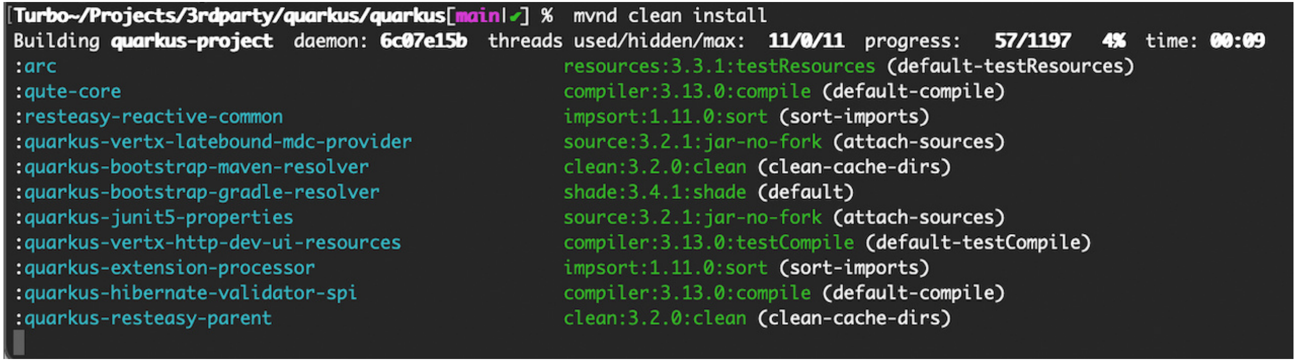
```
mvn clean install -DskipTests=true -Dgradle.cache.local.enabled=false -T6
```



1171 goals executed in 47 projects in 1m 4.900s

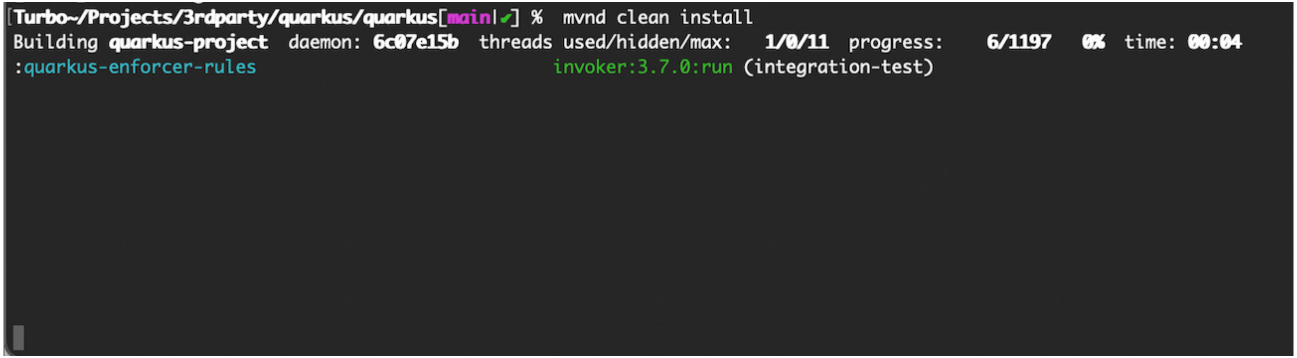
compiler:testCompile (default-testCompile) @ Netty/Common

mvnd visualization



```
[Turbo~/Projects/3rdparty/quarkus/quarkus[main|✓] %  mvnd clean install
Building quarkus-project  daemon: 6c07e15b  threads used/hidden/max:  11/0/11  progress:   57/1197   4%  time: 00:09
:arc                                                resources:3.3.1:testResources (default-testResources)
:qute-core                                          compiler:3.13.0:compile (default-compile)
:resteasy-reactive-common                           impsort:1.11.0:sort (sort-imports)
:quarkus-vertx-latebound-mdc-provider               source:3.2.1:jar-no-fork (attach-sources)
:quarkus-bootstrap-maven-resolver                   clean:3.2.0:clean (clean-cache-dirs)
:quarkus-bootstrap-gradle-resolver                  shade:3.4.1:shade (default)
:quarkus-junit5-properties                          source:3.2.1:jar-no-fork (attach-sources)
:quarkus-vertx-http-dev-ui-resources                compiler:3.13.0:testCompile (default-testCompile)
:quarkus-extension-processor                        impsort:1.11.0:sort (sort-imports)
:quarkus-hibernate-validator-spi                    compiler:3.13.0:compile (default-compile)
:quarkus-resteasy-parent                            clean:3.2.0:clean (clean-cache-dirs)
```

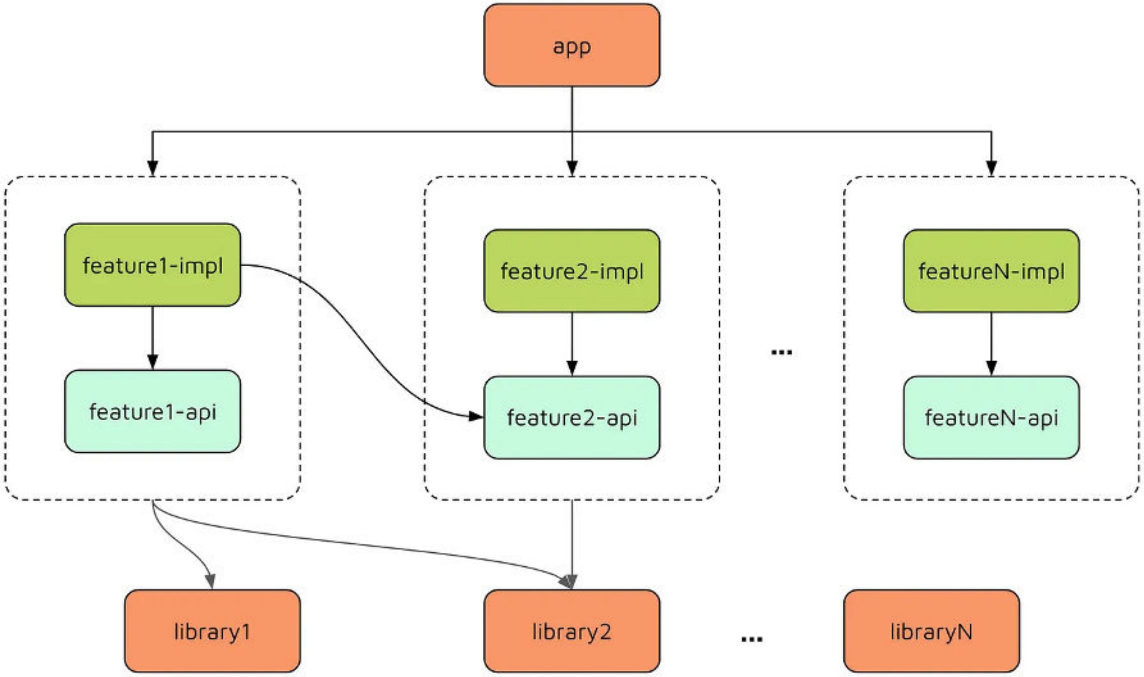
Don't be confused: while mvnd is parallel, mvn is not parallel by default



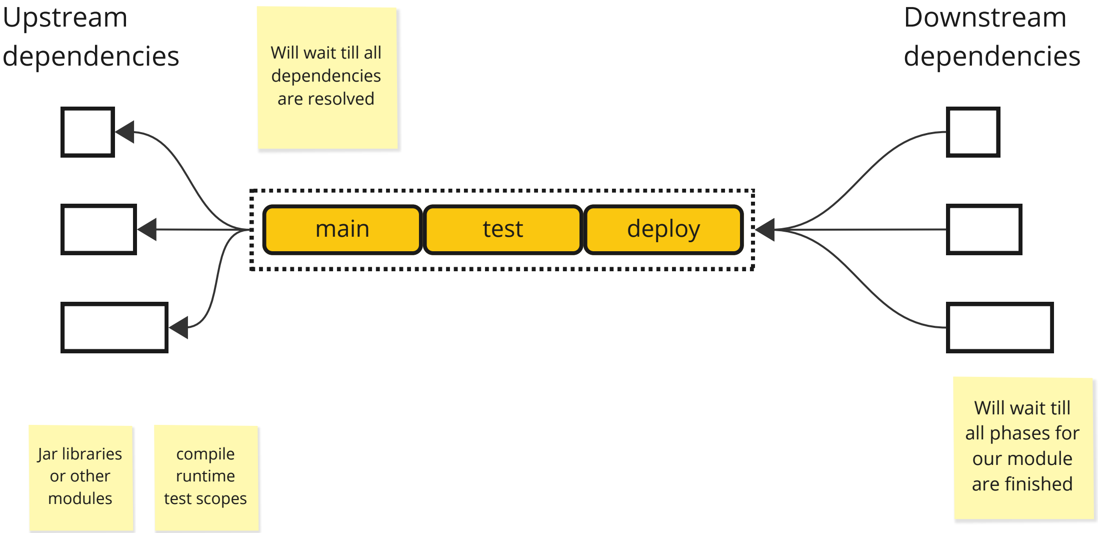
```
[Turbo~/Projects/3rdparty/quarkus/quarkus[main|✓] %  mvnd clean install
Building quarkus-project  daemon: 6c07e15b  threads used/hidden/max:   1/0/11  progress:    6/1197   0%  time: 00:04
:quarkus-enforcer-rules                            invoker:3.7.0:run (integration-test)
```

< That's a build bottleneck

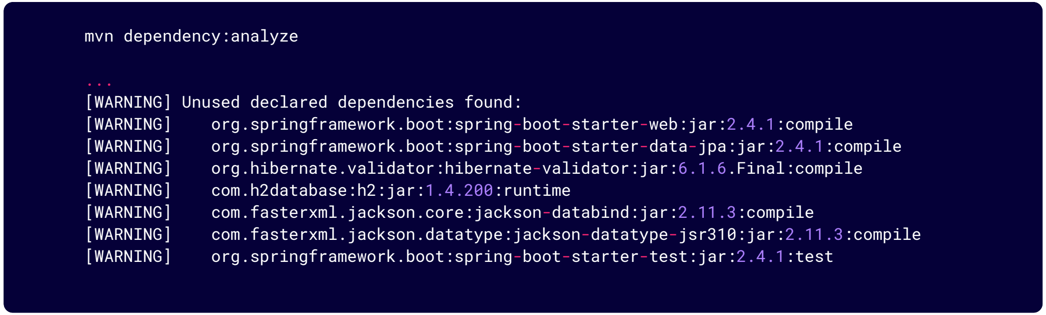
Modularization is a key point for optimization



How maven reactor works

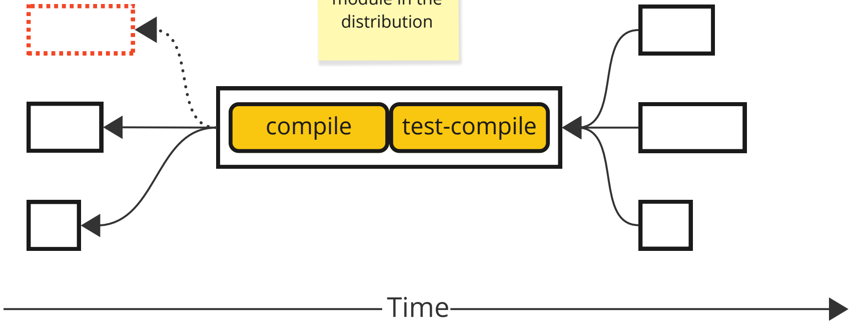

Upstream dependencies
Will wait till all dependencies are resolved
Downstream dependencies
main
test
deploy
Jar libraries or other modules
compile runtime test scopes
Will wait till all phases for our module are finished

Remove redundant dependencies



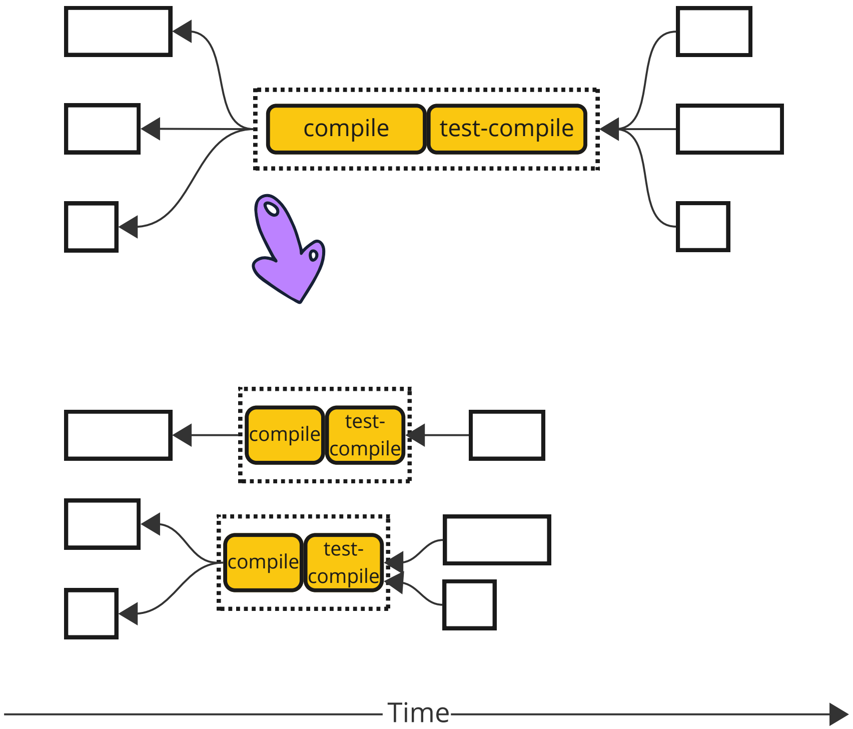
```
mvn dependency:analyze

...
[WARNING] Unused declared dependencies found:
[WARNING]    org.springframework.boot:spring-boot-starter-web:jar:2.4.1:compile
[WARNING]    org.springframework.boot:spring-boot-starter-data-jpa:jar:2.4.1:compile
[WARNING]    org.hibernate.validator:hibernate-validator:jar:6.1.6.Final:compile
[WARNING]    com.h2database:h2:jar:1.4.200:runtime
[WARNING]    com.fasterxml.jackson.core:jackson-databind:jar:2.11.3:compile
[WARNING]    com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.11.3:compile
[WARNING]    org.springframework.boot:spring-boot-starter-test:jar:2.4.1:test
```

Ensure you are not losing a module in the distribution

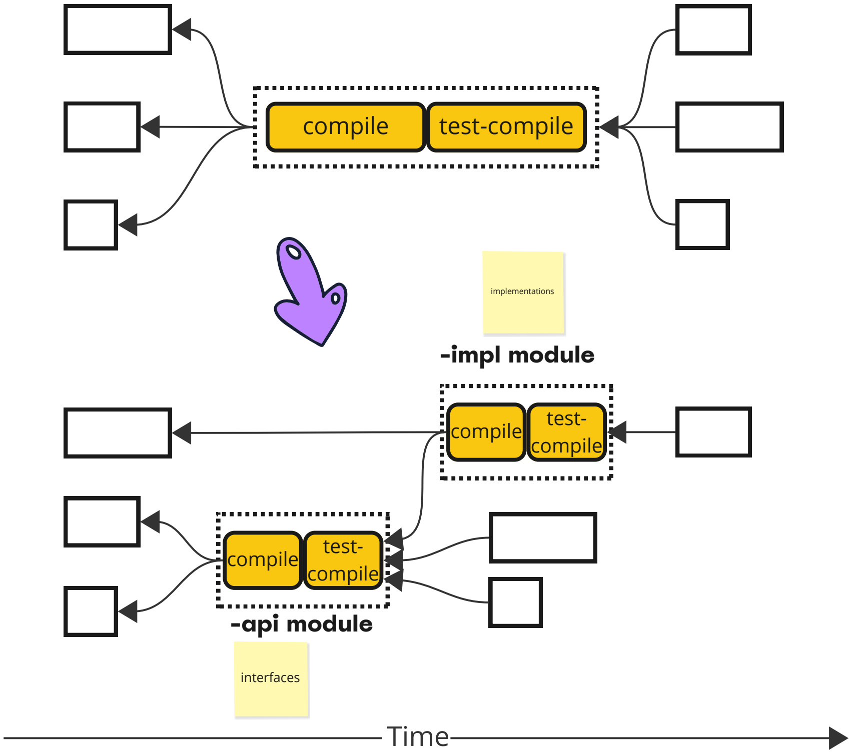


left-shifts on the timeline

Split large modules to smaller (independent)



Split large modules to smaller (dependent)



Possible with Dependency Injection or SPI

Extract code generation as jar dependency

protobuf

avro

jOOQ

OpenAPI (from YAML)

Shift to incremental compilation

Reduce log output

Log output consumes a lot of resources - e.g. tests debug info

Stop deploying redundant artifacts on each build

-sources.jar
-test-sources.jar
-javadoc.jar
-test-jar.jar

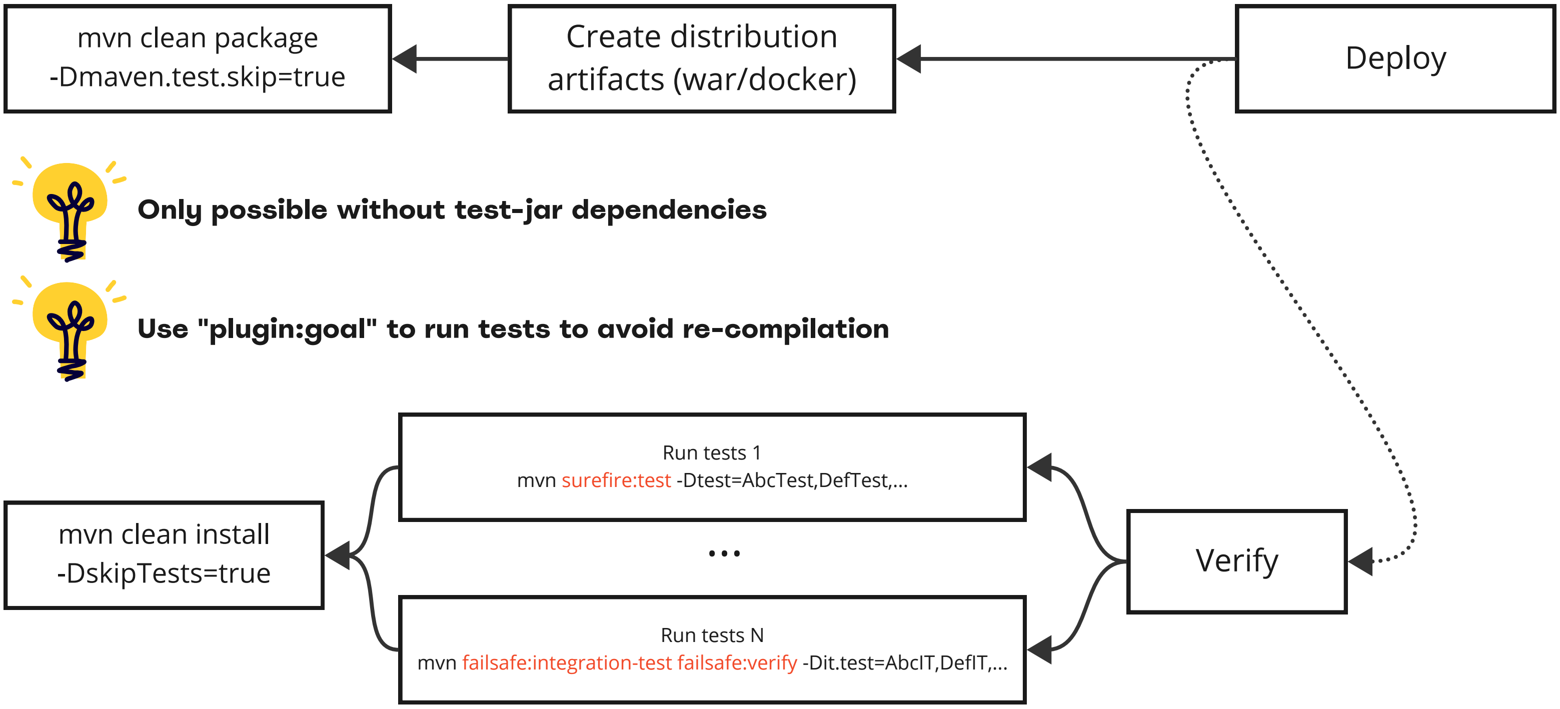
If you make a library, publish -sources

Remove dependencies to test-jar

```
...
    <dependencies>
        <dependency>
            <groupId>projects.pt.server</groupId>
            <artifactId>users</artifactId>
            <type>test-jar</type>
            <scope>test</scope>
        </dependency>
    </dependencies>
...
```

Imported classes should be rearranged to new "-test" modules

Parallel pipelines in CI/CD (faster deploy)



mvn clean package -Dmaven.test.skip=true

Create distribution artifacts (war/docker)

Deploy

mvn clean install -DskipTests=true

Run tests 1
mvn surefire:test -Dtest=AbcTest,DefTest,...

...

Run tests N
mvn failsafe:integration-test failsafe:verify -Dit.test=AbcIT,DefIT,...

Verify

Only possible without test-jar dependencies

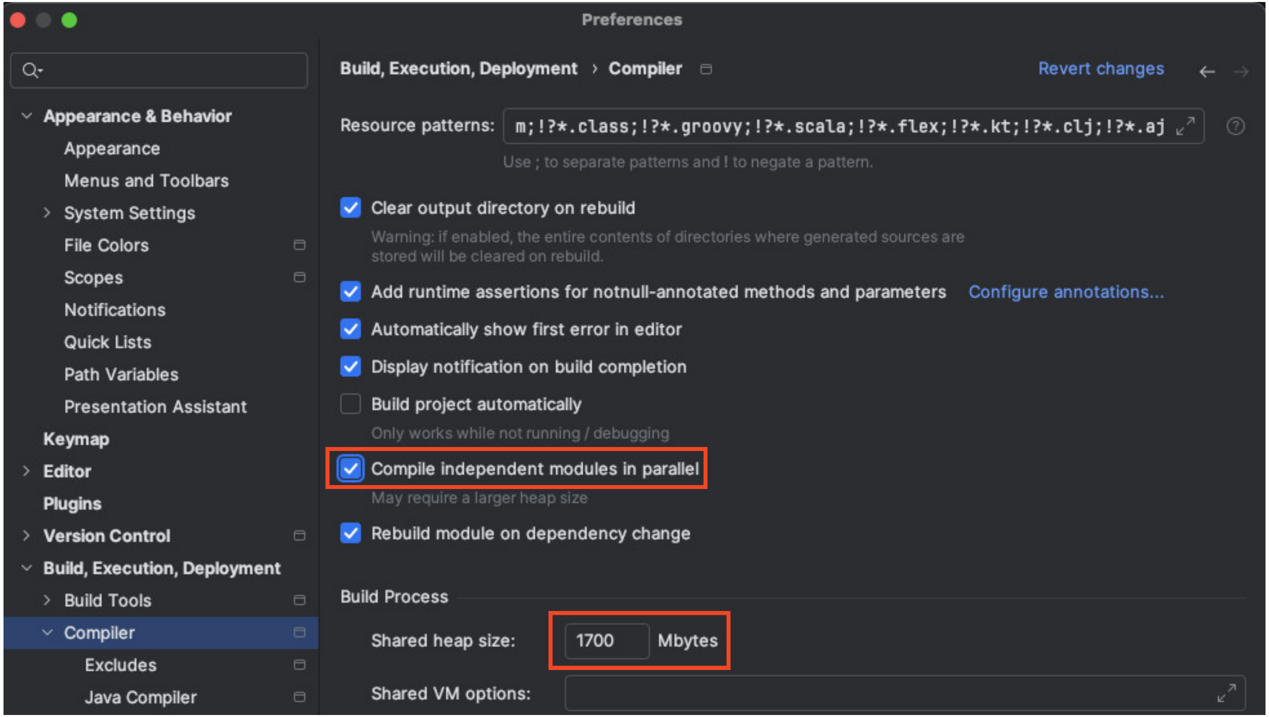
Use "plugin:goal" to run tests to avoid re-compilation

Persistent build agents

Reuse build agents with .m2 and build caches

Find a proper balance - not too long-living agents

IDEA: parallel compilation



Build, Execution, Deployment › Compiler

Resource patterns: `m;!?*.class;!?*.groovy;!?*.scala;!?*.flex;!?*.kt;!?*.clj;!?*.aj`

Use ; to separate patterns and ! to negate a pattern.

☑ Clear output directory on rebuild

Warning: if enabled, the entire contents of directories where generated sources are stored will be cleared on rebuild.

☑ Add runtime assertions for notnull-annotated methods and parameters

☑ Automatically show first error in editor

☑ Display notification on build completion

☐ Build project automatically

Only works while not running / debugging

☑ Compile independent modules in parallel

May require a larger heap size

☑ Rebuild module on dependency change

Build Process

Shared heap size: 1700 Mbytes

Shared VM options:

Make more operations optional (like code/report generation/verifications)

Hide them under profile

Migrate to Gradle

Use caches and other advantages of Gradle

More insights in Develocity build scans

Summary

- Check your JDK/Kotlin version
- Stop doing redundant operations
- Use cache
- Modularize, find bottlenecks
- Combine approaches to optimize average and worst cases

https://github.com/seregamorph/arch-maven-plugin

slides

SCAN ME