Project Beginnings

Martin Ledvinka

martin.ledvinka@fel.cvut.cz

Winter Term 2024

Contents

Project Management and Build

Apache Maven

- Software project management and comprehension tool
- Manage project dependencies, build, reporting, documentation
- Repository with libraries
  - Maven central at `maven.org` (Web UI at `http://central.sonatype.com`)
  - Possible to have own repository, see e.g. `https://kbss.felk.cvut.cz/m2repo`
  - Local repository – cache, in `${USER_HOME}/.m2`

POM

- *Project Object Model*
- `pom.xml` file
  - Central XML-based configuration of Maven projects
  - Hierarchical project identification
    - `groupId`
    - `artifactId`
    - `version`
  - Manage dependencies – `dependencies` section
  - Manage build process – `build` section – using plugins – `plugins` section

Directory Structure

- `src`
  - `/main`
    - `/java`
    - `/resources`
    - `/webapp`
  - `/test`
    - `/java`
    - `/resources`
- `pom.xml`

Project Build Phases

1. `validate` - validate the project structure and configuration
2. `compile` - compile the source code of the project
3. `test` - test the compiled source code using a suitable testing framework
4. `package` - take the compiled code and package it in its distributable format, such as a JAR
5. `verify` - run any checks on results of integration tests to ensure quality criteria are met
6. `install` - install the package into the local repository
7. `deploy` - copy the final package to the remote repository

Dependency Scopes

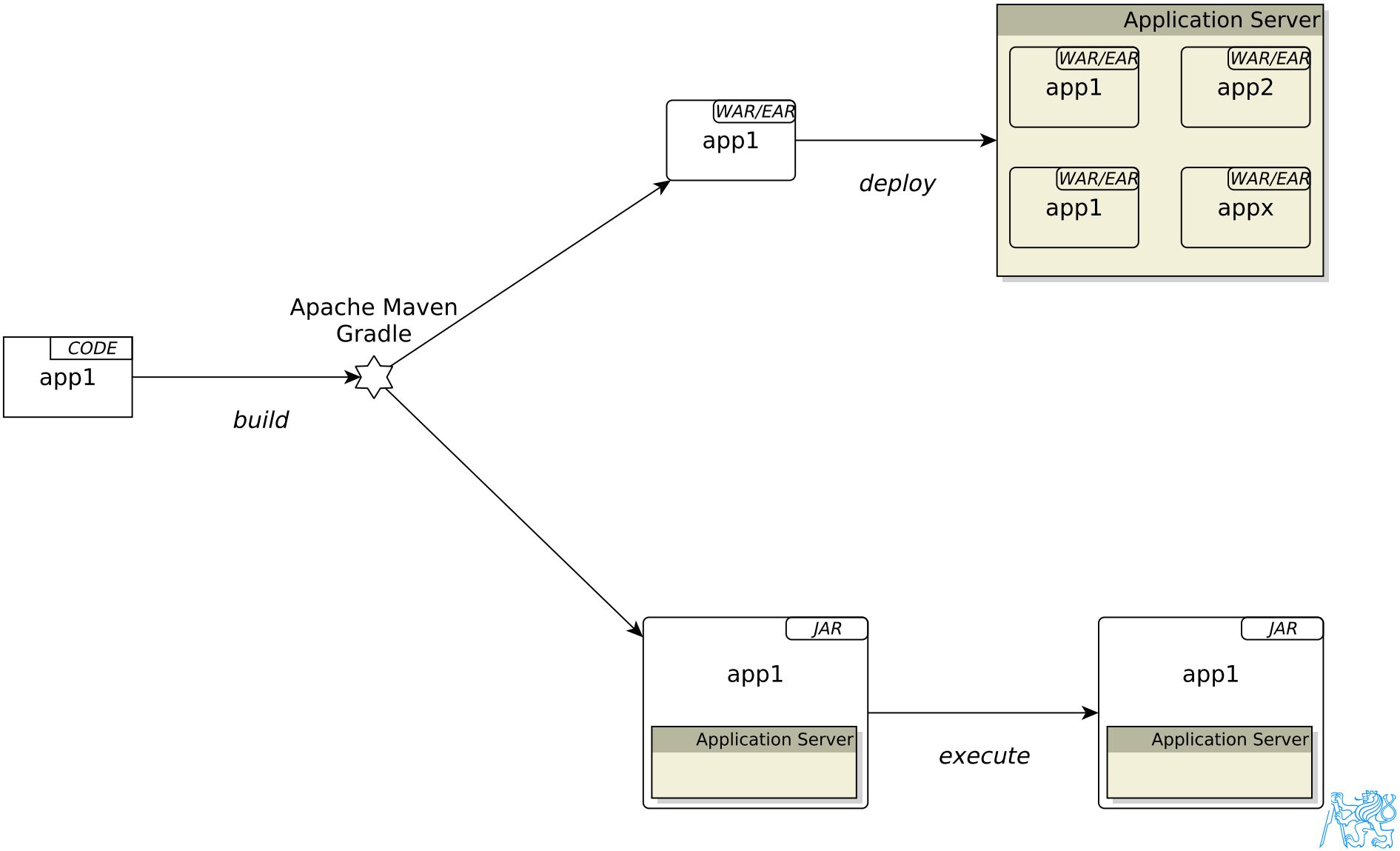
- `compile` – default, dependency available on classpath
- `provided` – expected to be provided at runtime – by JDK, application server etc.
- `runtime` – not required for compilation, but is for execution
- `test` – required for test compilation and execution
- `system` – similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
- `import` – used when specifying dependencies in parent projects

Gradle

Maven	**Gradle**
XML	Groovy
Maven repo	Maven repo
Plugins	Plugins, direct code
Recompile everything on change	Incremental build

Deployment

Deployment of Java Web Applications



WAR

- *Web Archive*
- Format of deployable Java web application artefacts
  - **EAR** for full-blown Jakarta (Java) EE artifacts

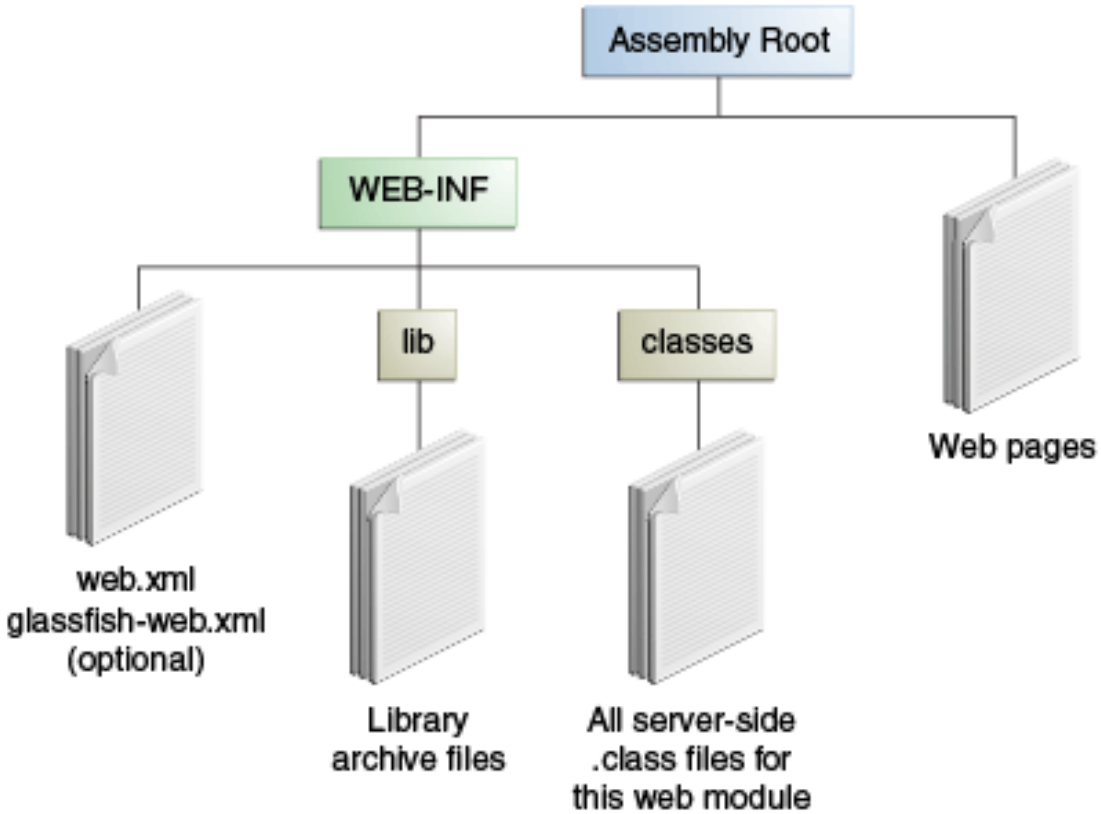


Figure: WAR structure. Source:
`https://docs.oracle.com/javaee/7/tutorial/packaging003.htm`

WAR cont.

- `web.xml` optional since Servlet API 3
  - All configuration can be done in source using Java + annotations
  - We won't be using it in our projects
- `WEB-INF` is not part of the public document tree of the application
  - Not accessible by clients
  - But accessible by servlet code – on classpath
  - Contains application code
- `lib` for required libraries, e.g., Spring, JDBC driver

Deployment

The following applies to Apache Tomcat!

- `webapps` folder for deployed web applications
- Can deploy exploded WAR (unpacked)
  - Tomcat will otherwise unpack WARs automatically
- Tomcat watches for changes in `webapps`
  - Copy into folder – *deploy*
  - Remove WAR from folder – *undeploy*
  - Application context
    - WAR file name
    - `META-INF/context.xml` in deployed WAR
    - `context.xml` in server configuration

Demo

- Demo of servlet application from lecture one to Tomcat
- Demo of servlet application from lecture one, deployment in IntelliJ IDEA via a Run configuration

JAR with Embedded Application Server

- Some libraries (Spring Boot, Quarkus) support creating an executable JAR file
- Such artifacts (typically) contain an embedded application server that runs the application in the JAR
- Deployment = executing the JAR

WAR vs JAR

WAR/EAR

- Application server manages multiple applications (or instances of one application)
- Application server can manage data sources
- More complex deployment, development

JAR

- Every instance runs in its own application server (and JVM)
- Simple deployment (execution), development
- Useful in containers (like Docker)

Tasks

Task – Demo

Inception of a Spring Boot project.

- Spring Boot by default packaged as JAR
- Use `spring-boot-starter-parent` Maven project parent to inherit dependencies easily
  - Various `spring-boot-starter-*` Maven projects pulling in groups of related dependencies
  - `spring-boot-starter-data-jpa` for JPA, transaction API
  - `spring-boot-starter-web` for Jackson, Spring Web, MVC and embedded Tomcat
- Build with `spring-boot-maven-plugin` to package dependecies into JAR automatically
- We can use the `SeminarTwoMain.java` class to check that the JAR can be executed

Task – 1 point

1. Fork project `https://gitlab.fel.cvut.cz/ear/b241-eshop`
2. Clone your fork of the B241-eshop project
3. Check out branch *b241-seminar-02-task*
4. Create a Maven/Gradle project using the `HelloWorld.java`, `Application.java`, `HelloWorldTest.java`, and `logback.xml` files

Acceptance Criteria

- Project has non-default groupId, artifactId, version, name and description
- Project can be packaged as **JAR** using Maven/Gradle
- Tests are run during build (and they pass)
- When the resulting JAR is executed, the servlet is accessible through a web browser
- Servlet access is logged based on the provided Logback configuration

Task – Hints

- Use Google or Maven central to find exact dependency identifiers
- `logback.xml` should go into `src/main/resources`
- You can use Spring initializr to bootstrap the project

Task – Bonus

- **1 bonus point**
- Create a similar configuration that builds a WAR that can be deployed to an application server
- Acceptance criteria: same as above, but build artifact is a deployable WAR

The End

Thank You

Resources

- http://maven.apache.org/guides/
- https://docs.oracle.com/javaee/7/tutorial/packaging003.htm
- https://spring.io/guides/gs/spring-boot/
- https://start.spring.io/