

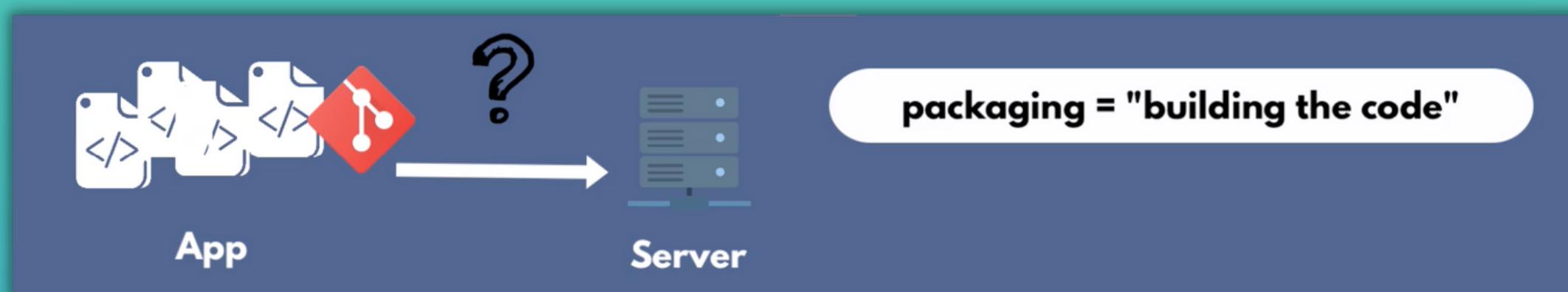


Build Tools & Package Manager

Key Takeaways

What are Build and Package Manager Tools? - 1

- Application needs to be deployed on a production server
- For that, we want to **package** application into a **single moveable file (artifact)**, also called "building the code"
- This is what a build tool or package manager tool does



What is an "artifact"?

- Includes application code and all its dependencies



artifact

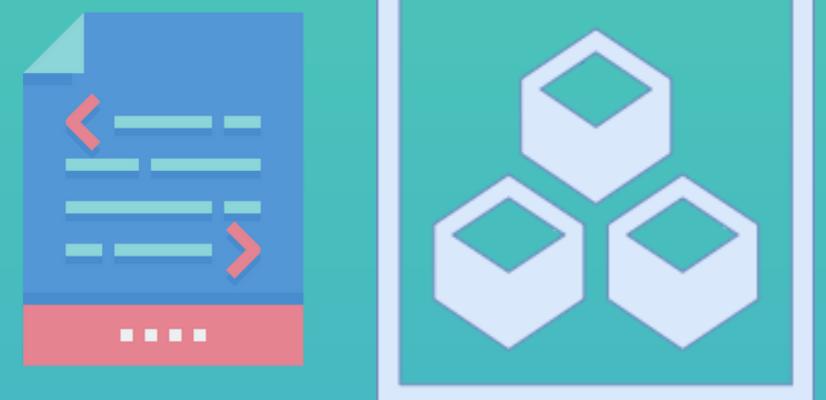
What are Build and Package Manager Tools? - 2

What does "building the code" mean?

- **Compiling** the code
- **Compressing** the code
- Package hundred of files **to 1 single file**

What is an "artifact repository"?

- Storage for artifacts
- To deploy it for example multiple times, have backup etc.
- Examples: Nexus, Jfrog Artifactory



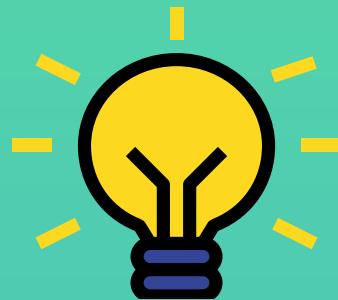
artifact repository

dev

test

production

Artifacts



Artifact files look different for each programming language



- JAR = Java Archive
- Includes **whole code plus dependencies**, like Spring framework, datetime libraries, pdf processing libraries



JAR or WAR file



- JavaScript's artifact can be a zip or tar file



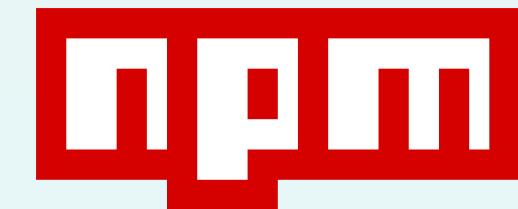
Zip or Tar file

Different Build Tools for different programming languages

Build tools we cover **in this module**:



- Java Build Tools: **Gradle and Maven**



- JavaScript Package Manager: **npm**

Installation and Setup of Projects - 1

- In this module you need to install following technologies and setup following projects:

Technologies

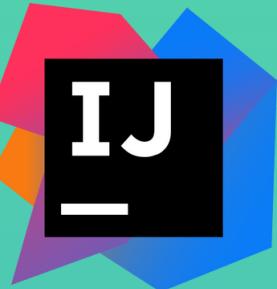


Projects

- **java gradle** application
- **java maven** application
- **nodejs** application

Installation and Setup of Projects - 2

- Download IntelliJ = **Code Editor**
- Install **OS Package Manager** (like Homebrew for Mac)
- Install Java, Maven, Node, npm



Clone Git Projects & Open in IntelliJ:

- Setup Java-**Maven** Project in IntelliJ
- Setup Java-**Gradle** Project in IntelliJ
- Setup Node.js-**npm** Project in IntelliJ

Build an Artifact



- With a Build Tool you can build the artifact
- Specific to the programming language
- Build Tools have **command line tool** and **commands to execute the tasks**

"Build" Process in Maven or Gradle:

- Installs dependencies
- Compiles and compresses your code



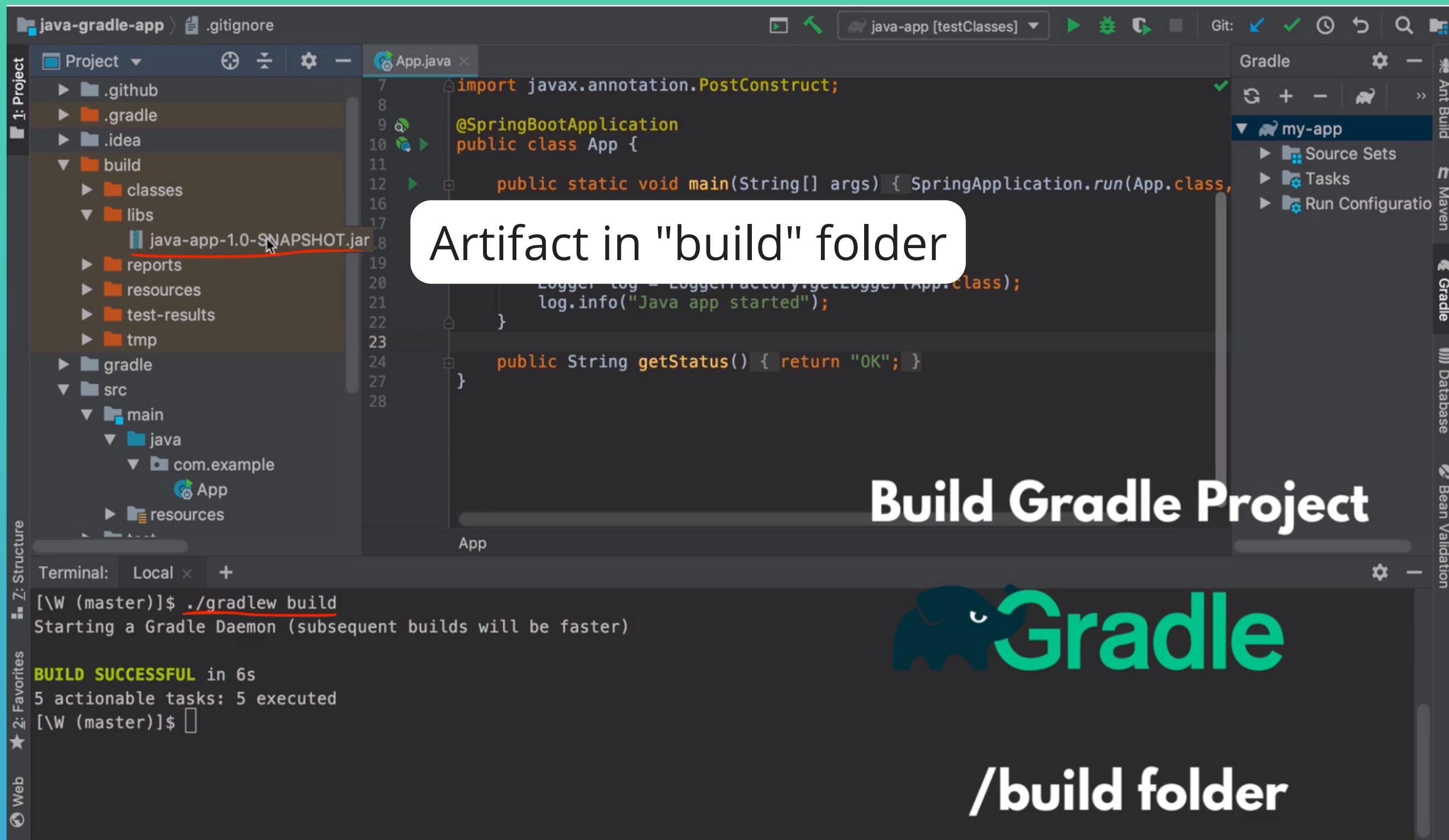
- Configuration in XML



- Configuration in Groovy

- Example Build Tools for Java: Gradle and Maven
- Artifact: JAR or WAR file

Building artifact for java-gradle project



Building artifact for java-maven project

The screenshot shows a Java-Maven project named "java-maven-app" in an IDE. The left panel displays the project structure and the `pom.xml` file. The right panel shows the file system structure and the terminal output.

Project Structure:

- Project: `java-maven-app`
- src/main/java/com/example/Application.java
- src/resources
- .gitignore
- pom.xml

pom.xml Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>java-maven-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>2.3.5.RELEASE</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

```

File System:

- Project: `java-maven-app`
- src/main/java/com/example/Application.java
- src/main/resources
- target
- classes
- maven-archiver
- maven-status
- java-maven-app-1.0-SNAPSHOT.jar
- .gitignore

Terminal Output:

```
[\W (master)]$ mvn install
[INFO] Scanning for projects...
[INFO] Total time:  1.989 s
[INFO] Finished at: 2020-11-22T11:26:31+01:00
[INFO] -----
```

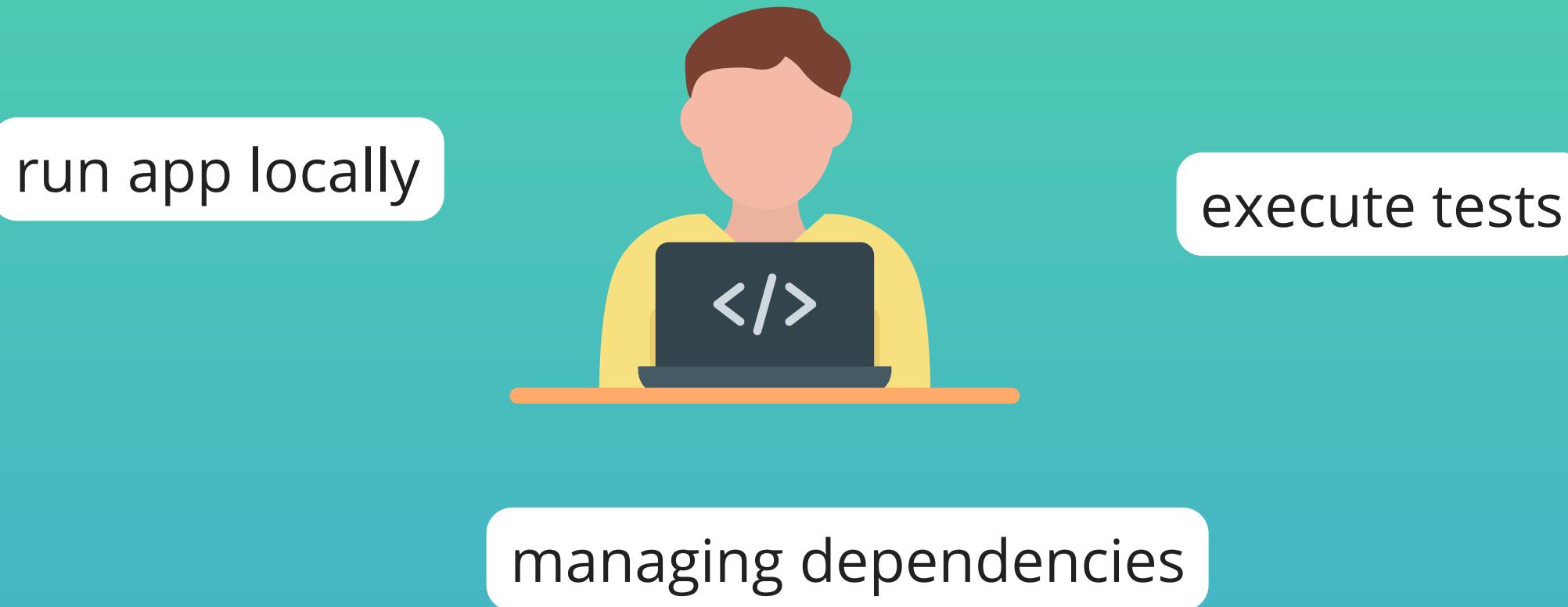
Artifact in "target" folder

Build Maven Project

Maver
/target folder

Build Tools for Software Development - 1

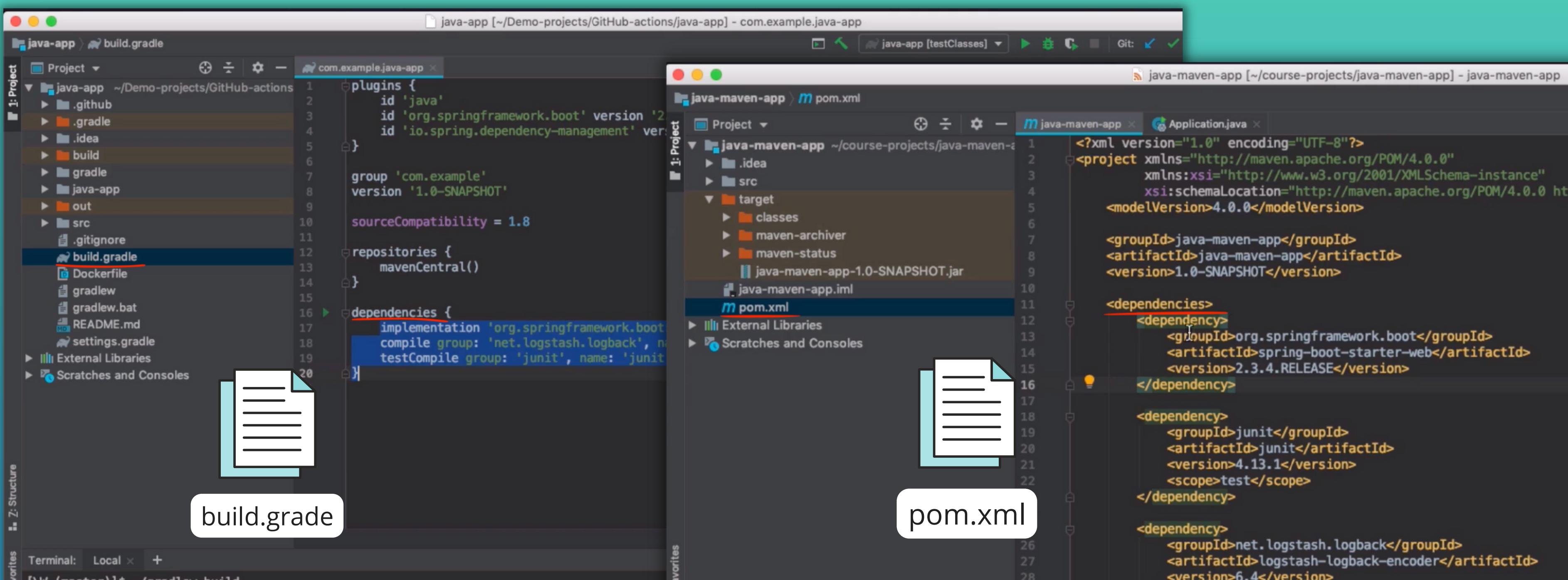
- Software developers need build tools **locally** when **developing the application**



- For example, a build tool is used to **manage the dependencies of a project**

Build Tools for Software Development - 2

- For that, build tools have a **dependencies file**:
- Whenever you need a new dependency, you can add it to the list



The image shows two side-by-side screenshots of IDE interfaces, likely IntelliJ IDEA, displaying build configuration files.

Left Screenshot (Gradle Project):

- The title bar says "java-app [~/Demo-projects/GitHub-actions/java-app] - com.example.java-app".
- The left sidebar shows project structure with files like .gradle, .idea, build, gradle, java-app, out, src, .gitignore, and build.gradle.
- The main editor shows the content of build.gradle:

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.3.4.RELEASE'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'

    group 'com.example'
    version '1.0-SNAPSHOT'

    sourceCompatibility = 1.8

    repositories {
        mavenCentral()
    }

    dependencies {
        implementation 'org.springframework.boot:spring-boot-starter-web'
        compile group: 'net.logstash.logback', name: 'logstash-logback-encoder', version: '6.4.1'
        testCompile group: 'junit', name: 'junit', version: '4.13.1'
    }
}
```

Right Screenshot (Maven Project):

 - The title bar says "java-maven-app [~/course-projects/java-maven-app] - java-maven-app".
 - The left sidebar shows project structure with files like .idea, src, target, classes, maven-archiver, maven-status, and java-maven-app-1.0-SNAPSHOT.jar.
 - The main editor shows the content of pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>java-maven-app</groupId>
    <artifactId>java-maven-app</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.3.4.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.1</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>net.logstash.logback</groupId>
            <artifactId>logstash-logback-encoder</artifactId>
            <version>6.4.1</version>
        </dependency>
    </dependencies>

```

Annotations:

 - A large white box labeled "build.gradle" is positioned over the left screenshot.
 - A large white box labeled "pom.xml" is positioned over the right screenshot.

Build Tools for Software Development - 3

- You can find all Java Libraries in Maven Central Repository:

The screenshot shows the Maven Repository website at mvnrepository.com/artifact/junit/junit/4.13.1. The page displays detailed information about the JUnit 4.13.1 artifact, including its license (EPL 1.0), categories (Testing Frameworks), organization (JUnit), homepage (<http://junit.org>), and the fact that it is used by 102,470 artifacts. A graph on the left shows the growth of indexed artifacts from 2006 to 2018. Below the artifact details, there is a code snippet of the dependency declaration, which is highlighted with a red circle. The snippet is as follows:

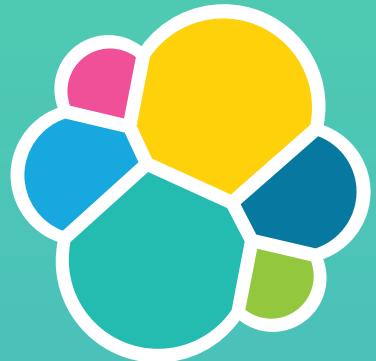
```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
    <scope>test</scope>
</dependency>
```

At the bottom of the code block, there is a checked checkbox labeled "Include comment with link to declaration" and a message "Copied to clipboard!".

Build Tools for Software Development - 4

Need a library for ElasticSearch database connection?

1. Find a dependency with name and version
2. You add it to dependencies file (e.g. pom.xml)
3. Dependency gets downloaded locally (e.g. local maven repo)



Run a Java Application

Locally, for example to test:

- Locate Jar file and execute:

```
java -jar <name of jar file>
```

On a **deployment server**:

- Copy the jar file to server, where application should run and execute "java -jar" command

The screenshot shows a Java IDE interface. On the left, there's a tree view of a Maven project named 'Application'. It includes a 'resources' folder, a 'target' folder containing 'classes', 'maven-archiver', and 'maven-status' subfolders, and a 'java-maven-app-1.0-SNAPSHOT.jar' file. Below these are '.gitignore', 'pom.xml', 'External Libraries', and 'Scratches and Consoles'. A large white curved arrow points from the 'java -jar' command text above to the 'java-maven-app-1.0-SNAPSHOT.jar' file in the project tree. To the right of the tree is a vertical list of numbers from 9 to 25. To the far right, a portion of the 'pom.xml' file is visible, showing XML code for a Maven build configuration.

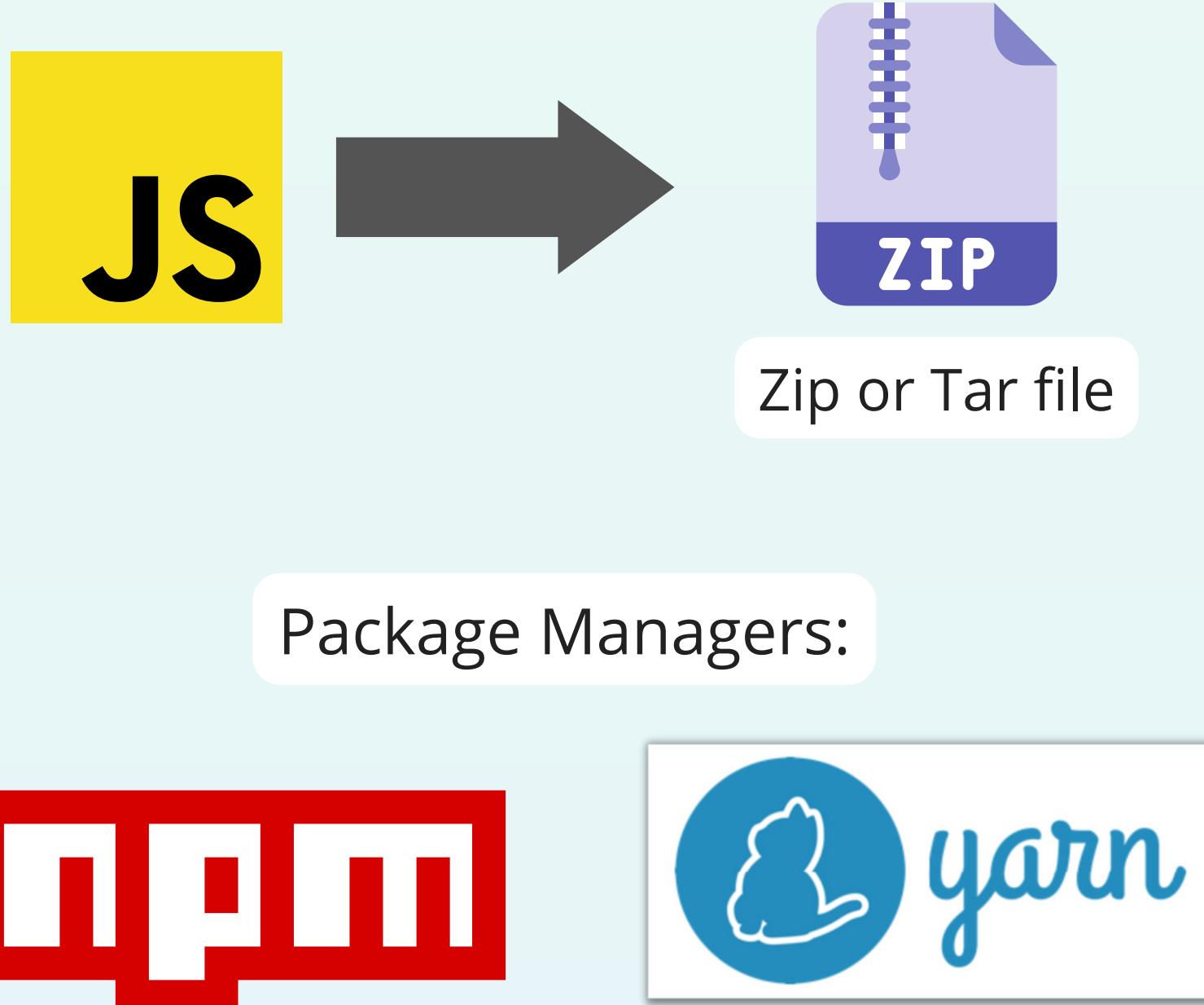
```
<version>1.0-SNAPSHOT</ve  
<build>  
  <plugins>  
    <plugin>  
      <groupId>org.  
      <artifactId>s  
      <version>2.3.  
      <executions>  
        <executio  
          <goal>  
            </goal  
          </executio  
        </executio  
      </executions>  
    </plugin>  
  </plugins>  
</build>  
<project> > build > plugins > plugin > e
```

Terminal: Local × +

```
(master]$ java -jar target/java-maven-app-1.0-SNAPSHOT.jar
```

Build JavaScript applications - 1

- Artifact of a JavaScript application is e.g. a **zip or tar file**
- But no special artifact type defined
- In JS we have **package managers** and NOT build tools
- For JS we have npm and yarn
- **package.json** file for dependencies
- Package manager install dependencies, but not used for transpiling JS code



Build JavaScript applications - 2

- JavaScript libraries can be found in **npm repository**

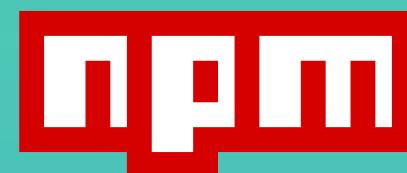
This screenshot shows the npm search interface. The search bar at the top contains the query 'mongodb'. Below the search bar, it says '6341 packages found'. On the left, there are filters for 'Sort Packages' (Optimal), 'Popularity', 'Quality', and 'Maintenance'. The main list includes:

- mongodb** (exact match) - The official MongoDB driver for Node.js. Published by mbroadst, version 3.6.2, a month ago. Tags: mongodb, driver, official.
- mongoose** - Mongoose MongoDB ODM. Published by vkarpov15, version 5.10.9, 12 days ago. Tags: mongodb, document, model, schema, database, odm, data, datastore, query, nosql.
- bson**

This screenshot shows the npm package page for 'mongodb'. The URL is 'npmjs.com/package/mongodb'. The page header includes the npm logo and a search bar. It features a summary for version 3.6.2, which is public and was published a month ago. Key statistics include 6 dependencies, 8,624 dependents, and weekly downloads of 2,036,164. The 'Description' section states: 'The official MongoDB driver for Node.js. Provides a high-level API on top of `mongodb-core` that is meant for end users.' A note at the bottom says: 'NOTE: v3.x was recently released with breaking API changes. You can find a list of changes [here](#)'.

Build JavaScript applications - 3

Command Line Tool - npm



- **npm install:** installs the dependencies
- **npm start:** start the application
- **npm stop:** stop the application
- **npm test:** run the tests
- **npm publish:** publish the artifact

What does the zip/tar file include?

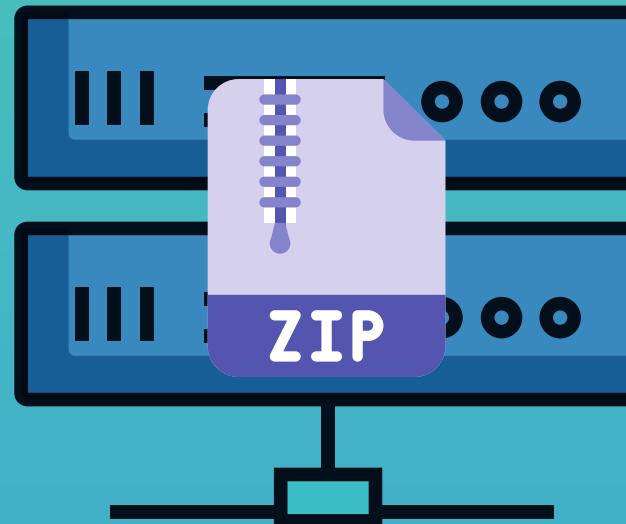
- application code, but **NOT the dependencies**

Build JavaScript applications - 4

Run JavaScript application on server:

1. Copy Zip/Tar file to server
2. Unpack zip/tar
- 3. Install dependencies**
4. Run the application

- Zip/Tar file includes the application code, but **NOT the dependencies**



Flexible JavaScript - 1



JavaScript world is **more flexible** and less
standardized compared to Java

If application consists of different programming languages:

Frontend: React.js (JS Library)

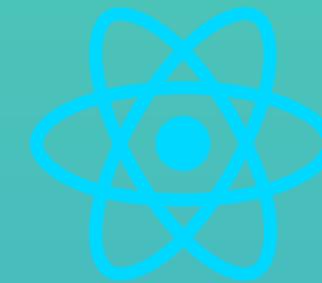
Backend: Java, Node.js, Python etc.

- You can package frontend and backend code separately
- Or in a common artifact file

Flexible JavaScript - 2

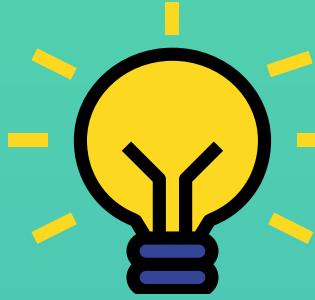
If application consists of JavaScript in Frontend and Backend:

Frontend: React.js (JS Library) **Backend:** Node.js (JS Library)



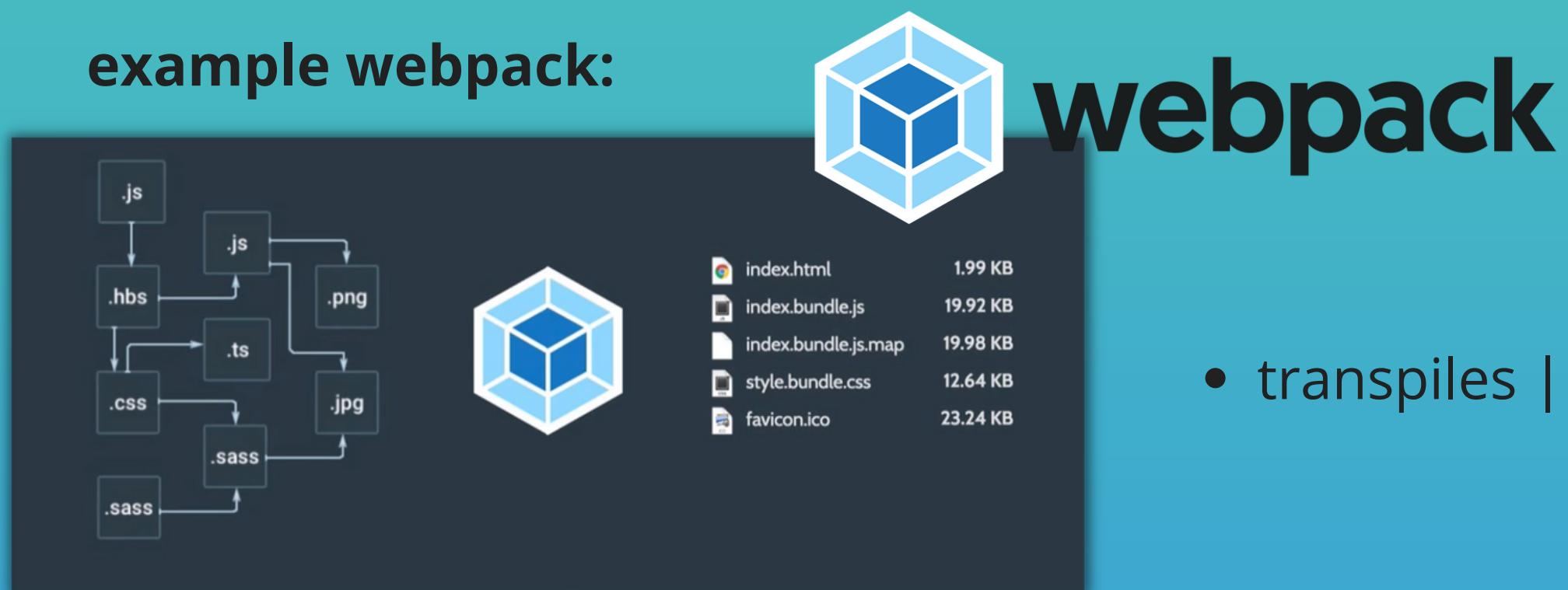
- You can have a **separate package.json file** for frontend and backend
- Or have a **common package.json file**

Package Frontend Code - 1

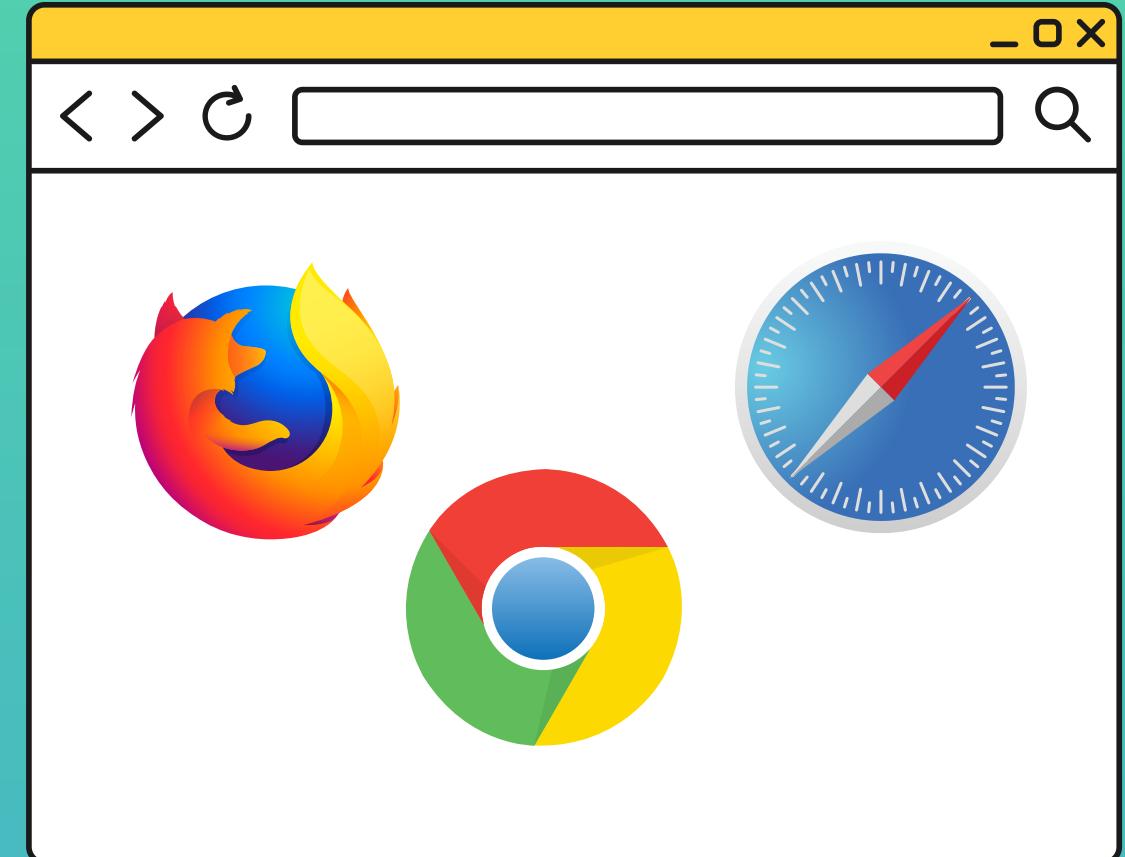


Frontend Code runs in the browser.
Browser don't support latest JS versions or other fancy code decorations, like JSX in React.js

- That's why frontend/React Code needs to be **transpiled** into browser compatible code
- Code needs to be **compressed/minified**
- There are separate tools for that: **Build Tools/Bundler**, for example **webpack**:



- transpiles | minifies | bundles | compresses the code



Package Frontend Code - 2

-  **Bundle** frontend code **with webpack**
-  **Manage dependencies** with **npm or yarn**
-  **Package** everything into a **WAR file**

Build Tools for other programming languages

- **Java**: maven | gradle
- **JavaScript**: npm | yarn | webpack
- **Python**: pip
- **C/C++**: conan
- **C#**: NuGet
- **Golang**: dep
- **Ruby**: RubyGems



But, concepts very similar to other
programming languages

Pattern in all these tools:

1. dependency file

package.json

pom.xml

build.gradle

2. repository for dependencies

3. command line tool, to:

test

start the app

build app

publish app

4. package managers

gradle

npm

maven

yarn

pip

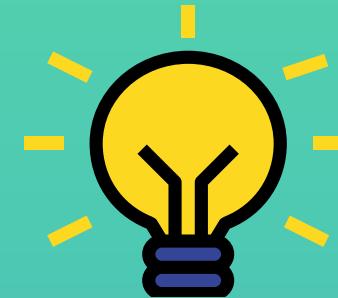
dep

Publish an artifact

1. Build jar or zip package
2. **Push to artifact repository** to save those packages for later use or for downloading on a remote server

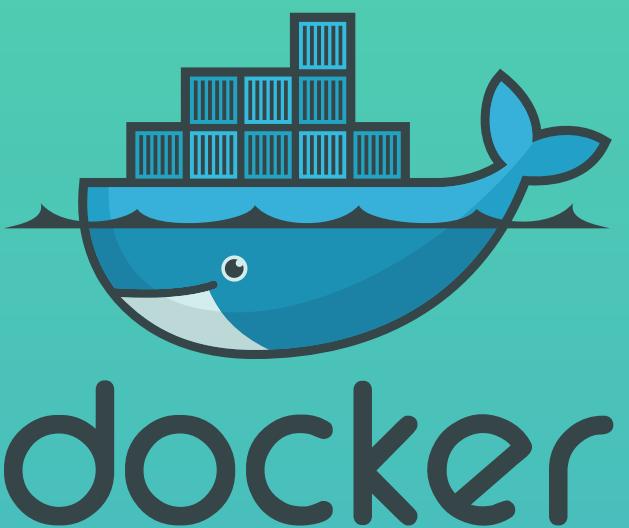


3. Then you can download (curl, wget) it anywhere



Change in how we use artifacts

- We don't keep jar or zip files, because we have Docker
- We don't build them locally, because we have Jenkins and other Build Automation Tools



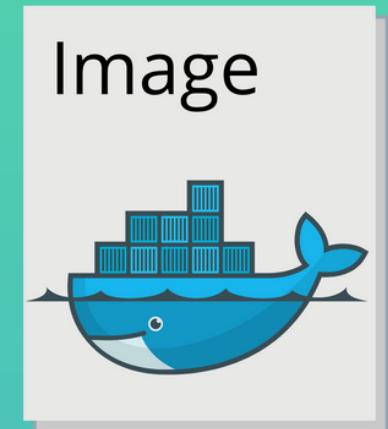
Jenkins

Build Tools and Docker - 1



No need to build and move different artifact types (e.g. Jar, War, Zip)

Just 1 artifact type: Docker Image



- We build those Docker images from the applications



No need for a repository for each file type

Just 1 Docker Image



No need to install dependencies on the server!

Execute install command inside Docker Image

Build Tools and Docker - 2



Docker makes it easier

Docker Image is an artifact

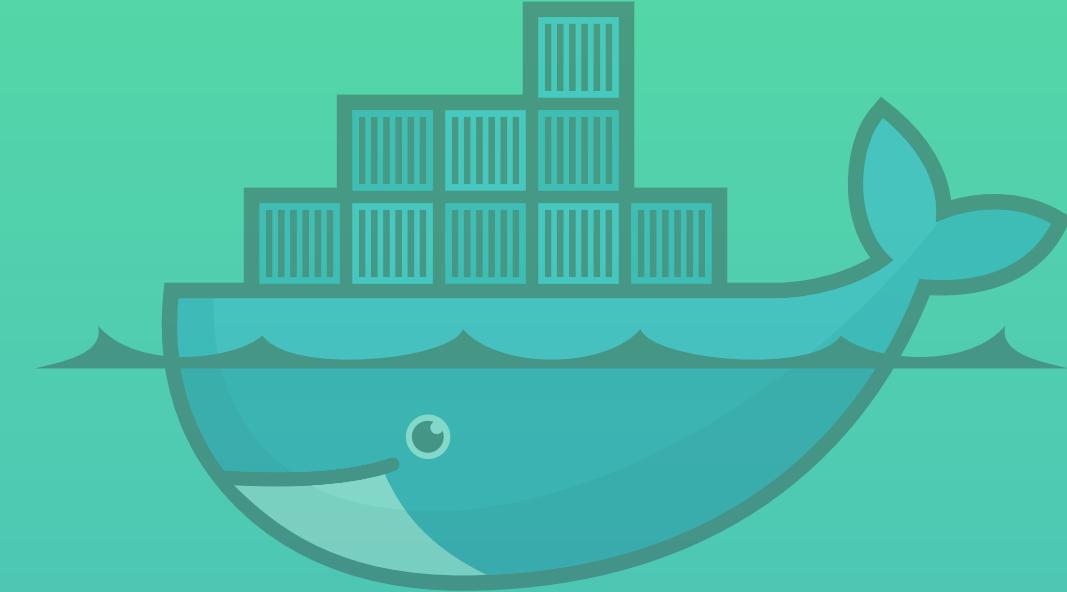


Docker Image is an alternative for all other artifact types

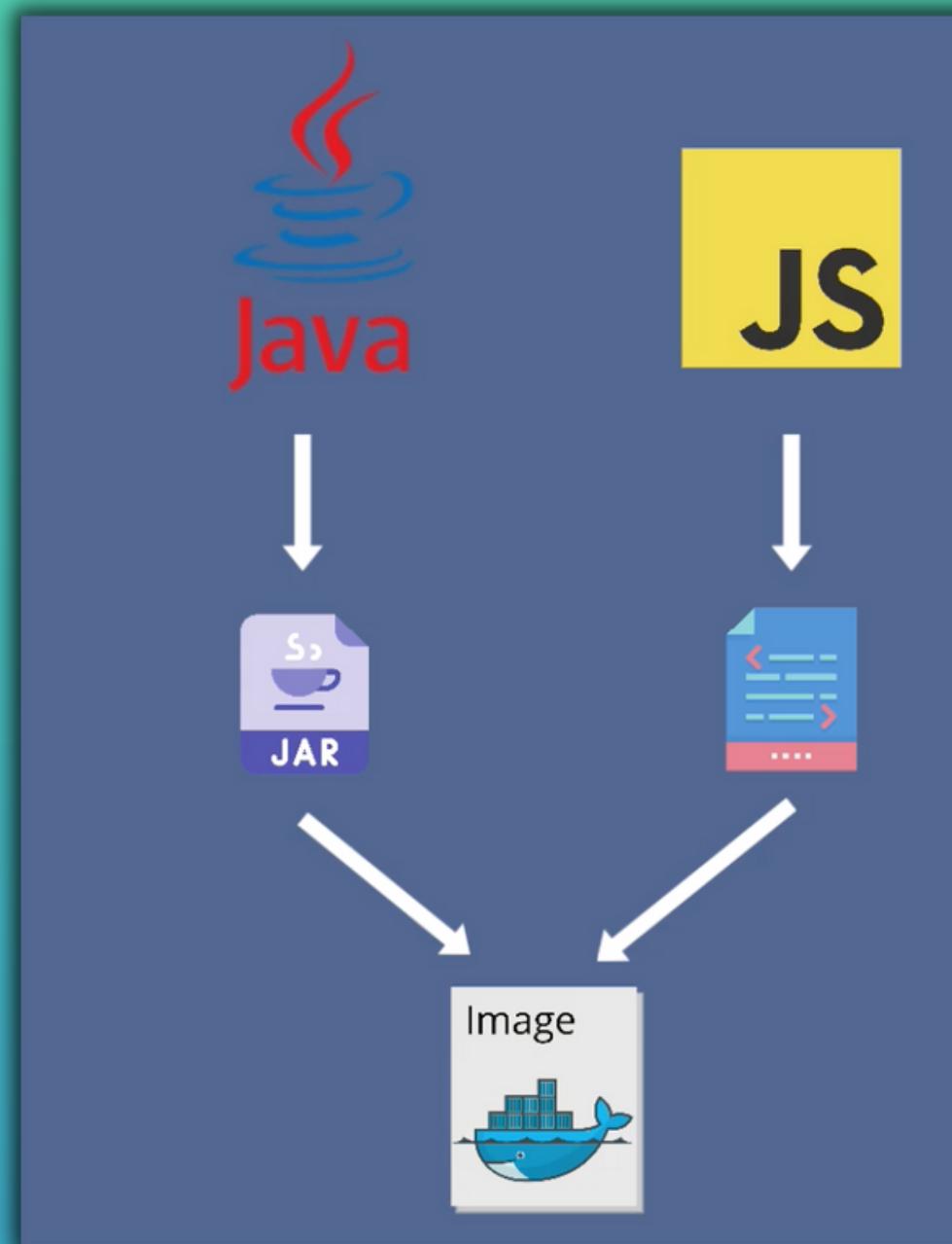


You don't need to install npm or java on the server

Execute everything in the Image



Build Tools and Docker - 3



A screenshot of a code editor showing a Dockerfile for a Java application named 'java-app'. The Dockerfile specifies:

```
FROM openjdk:8-jre-alpine
EXPOSE 8080
COPY ./build/libs/java-app-1.0-SNAPSHOT.jar /usr/app/
WORKDIR /usr/app
ENTRYPOINT ["java", "-jar", "java-app-1.0-SNAPSHOT.jar"]
```

The project structure on the left shows files like .github, .gradle, .idea, build, classes, libs, reports, resources, test-results, tmp, gradle, java-app, src, .gitignore, build.gradle, Dockerfile, and gradlew.

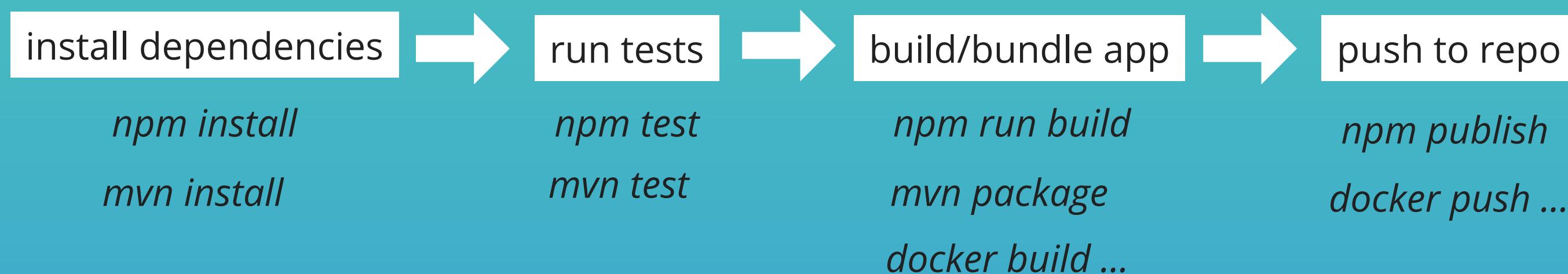
- You still need to build the application!



More about Docker in later Module!

Why should you know these Build Tools as a DevOps engineer?

- **Help developers** building the application, because you know where and how it will run on deployment servers
- You need to **configure the build automation tool** or CI/CD Pipeline, like execute tests on the build servers, build and package into Docker Image, run the application on server



So, you don't execute anything locally