Build Tools & Package Manager

Key Takeaways

TECHWORLD WITH NANA

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

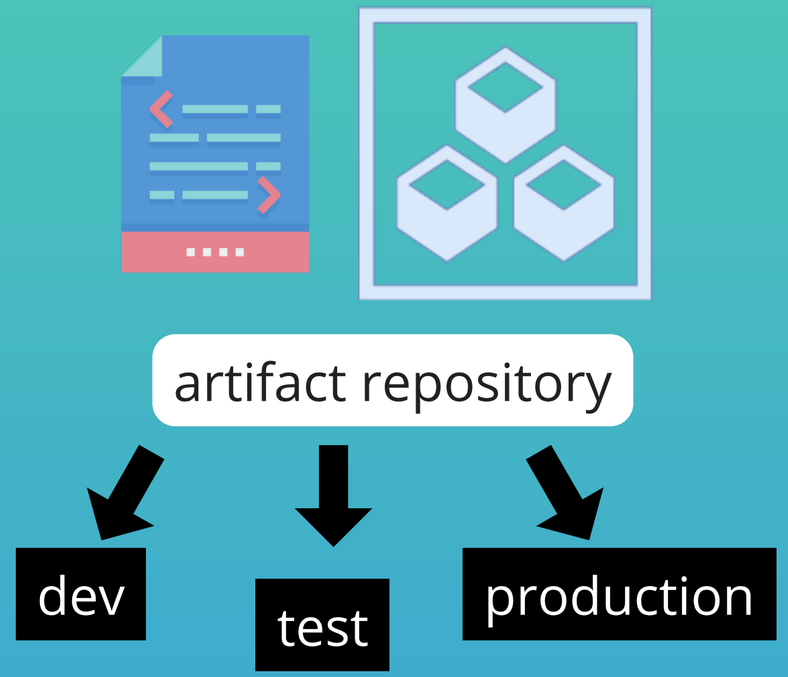
What are Build and Package Manager Tools? - 2

What does "building the code" mean?

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

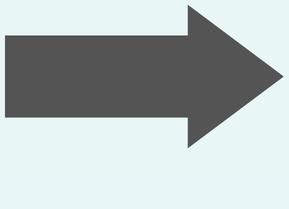
What is an "artifact repository"?

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



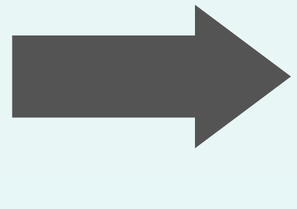
Artifacts

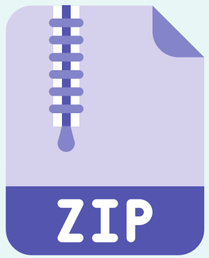
Artifact files look different for each programming language



JAR or WAR file

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



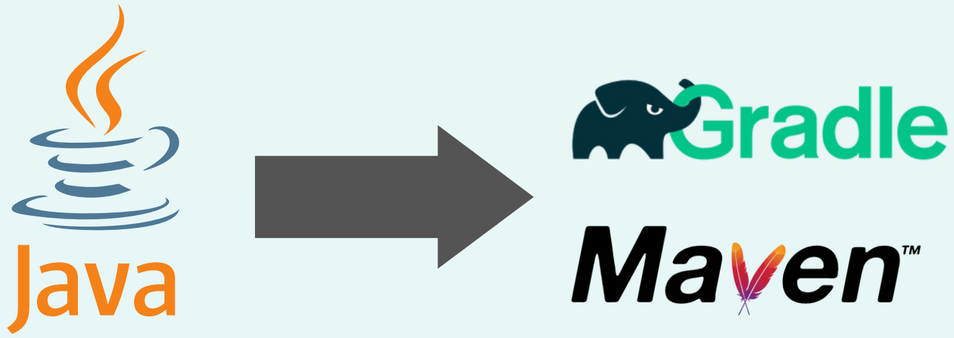


Zip or Tar file

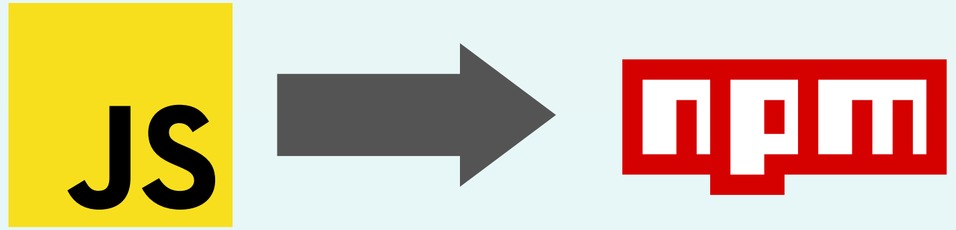
- JavaScript's artifact can be a zip or tar file

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

Gradle

- Configuration in XML

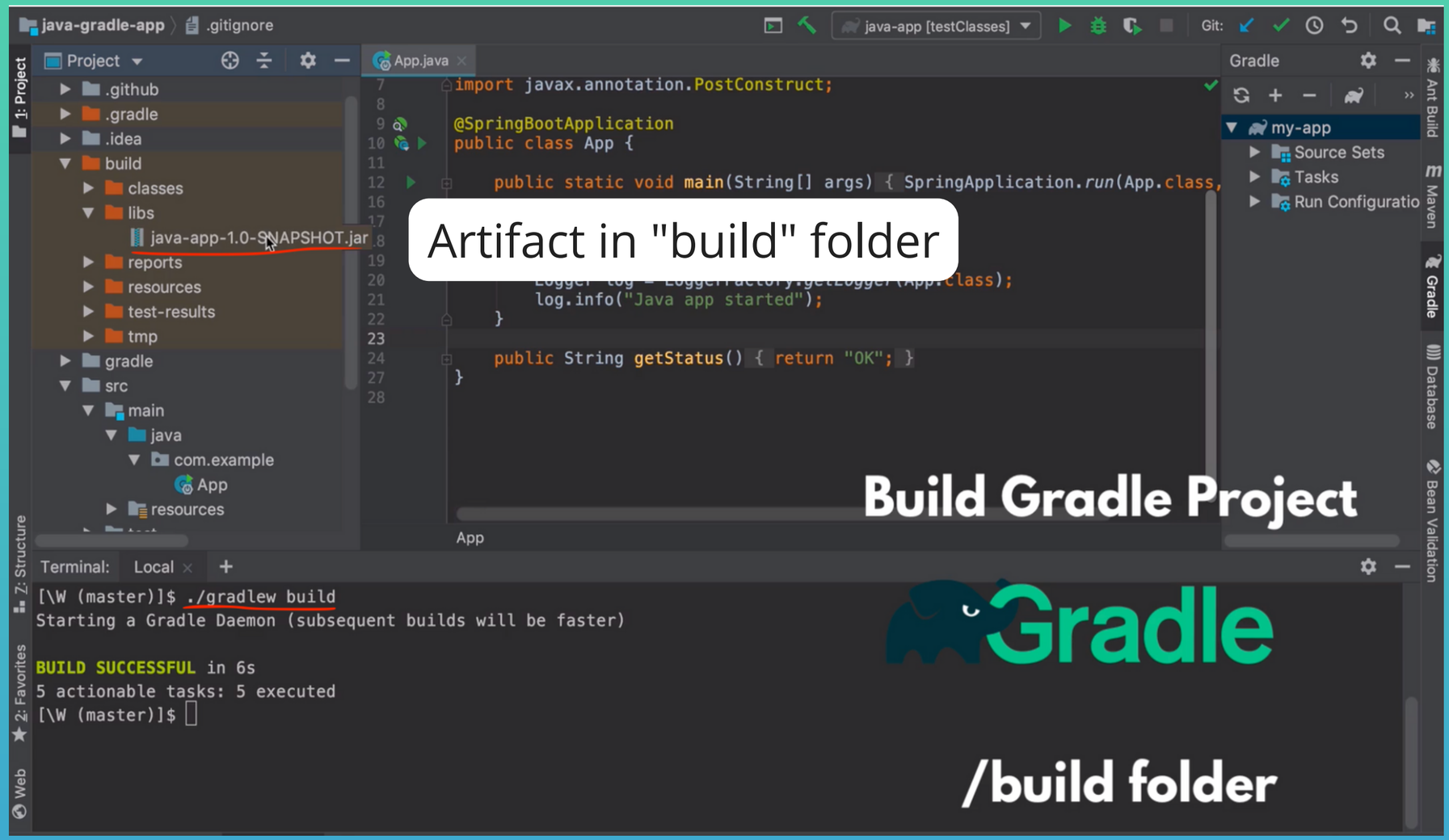
Maven™

- Configuration in Groovy

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

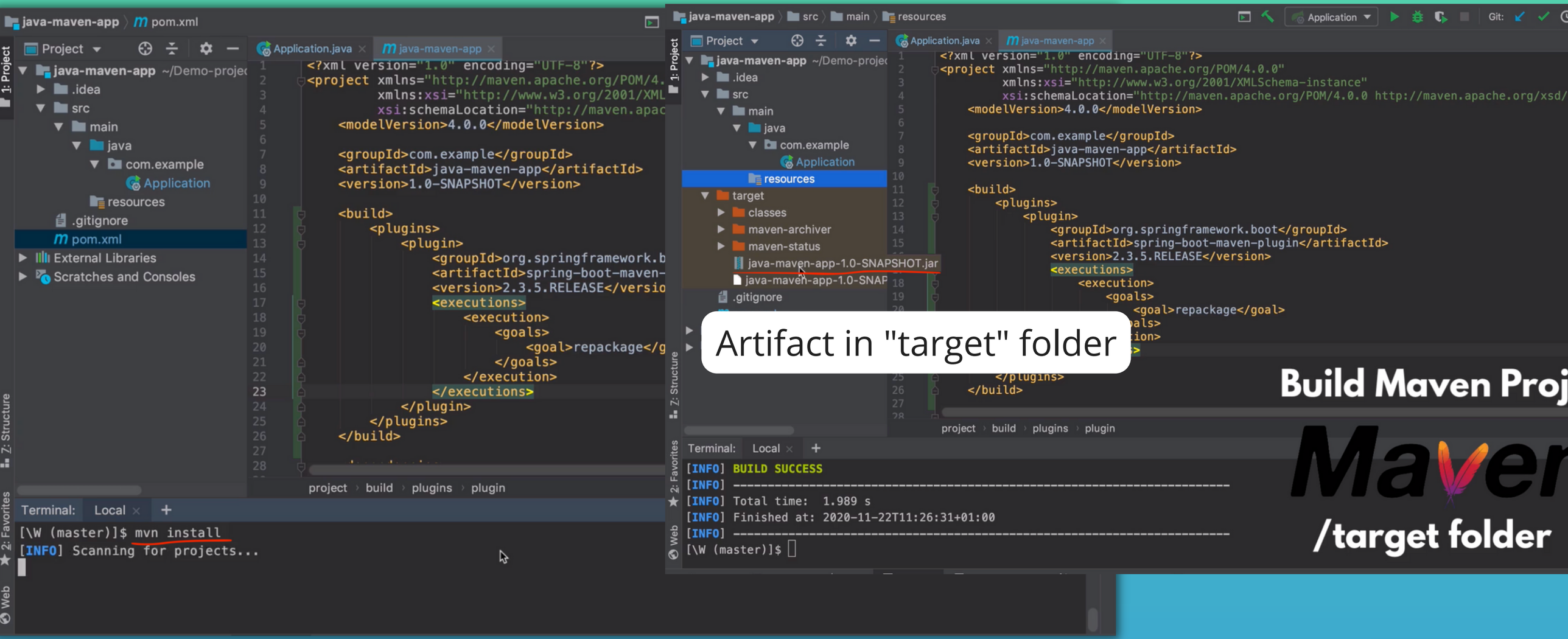
TECHWORLD WITH NANA

Building artifact for java-gradle project



TECHWORLD WITH NANA

Building artifact for java-maven project



Artifact in "target" folder

Build Maven Proj

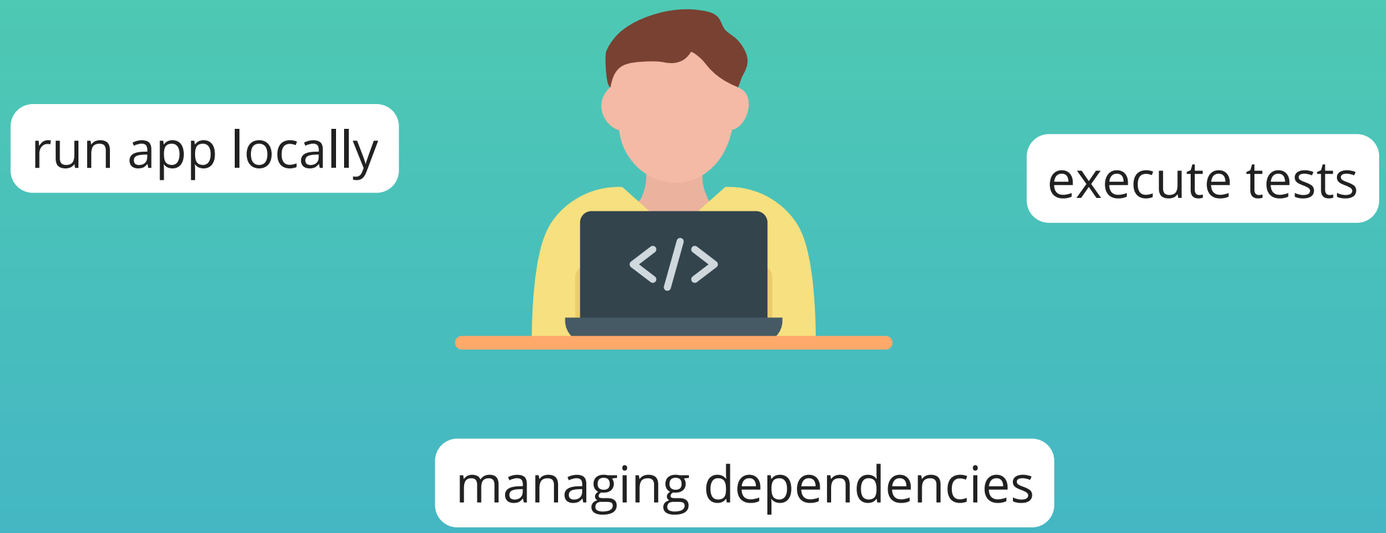
Maven

/target folder

TECHWORLD WITH NANA

Build Tools for Software Development - 1

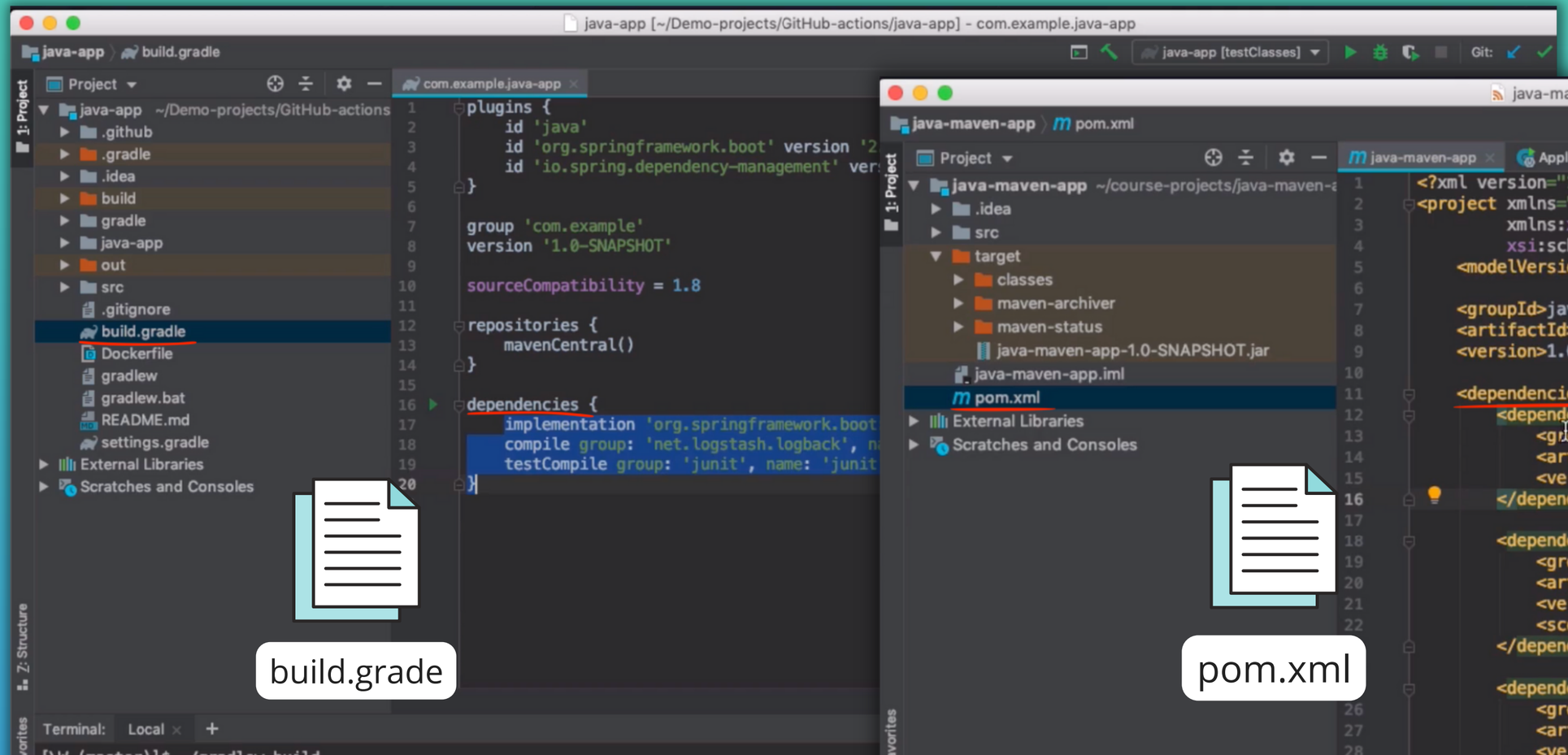
- Software developers need build tools **locally** when **developing the application**



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

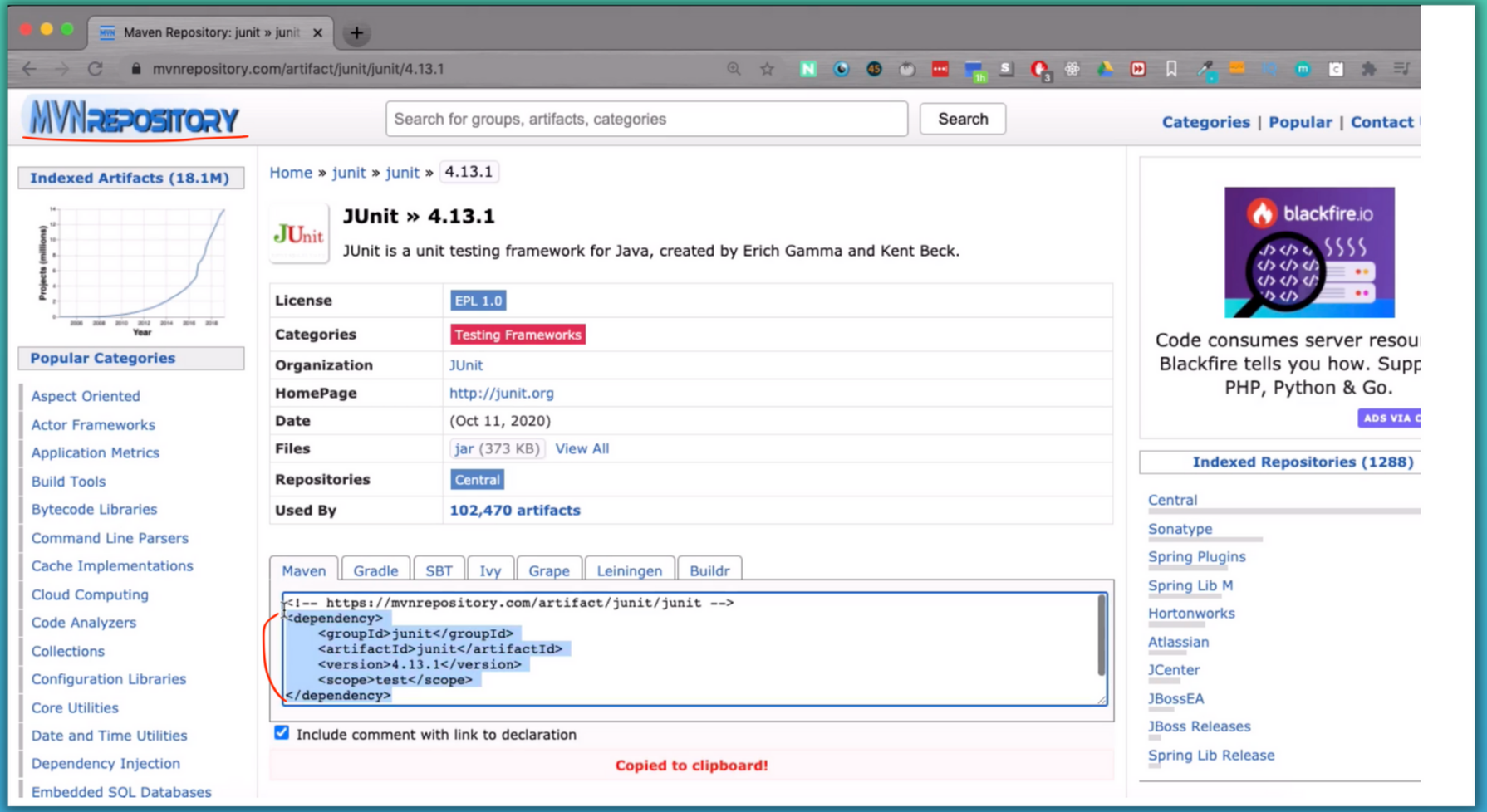
Build Tools for Software Development - 2

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



Build Tools for Software Development - 3

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



Build Tools for Software Development - 4

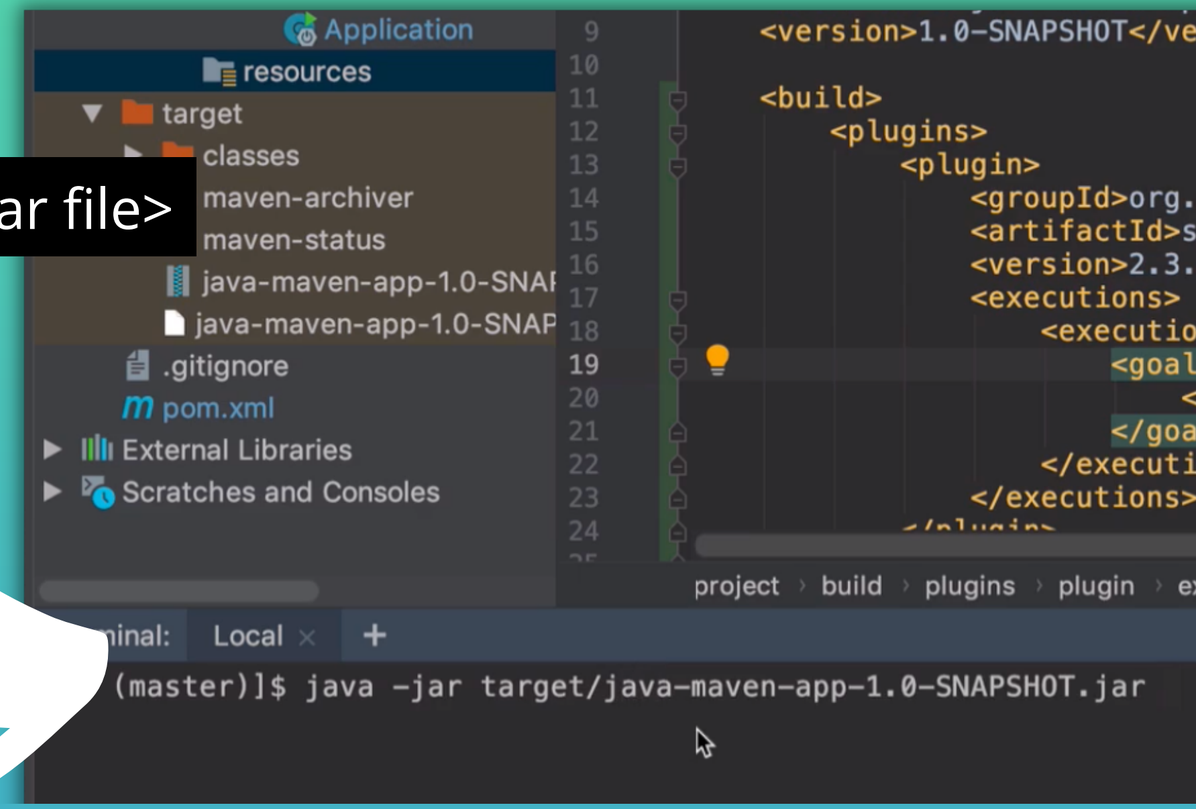
Need a library for ElasticSearch database connection?

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>`



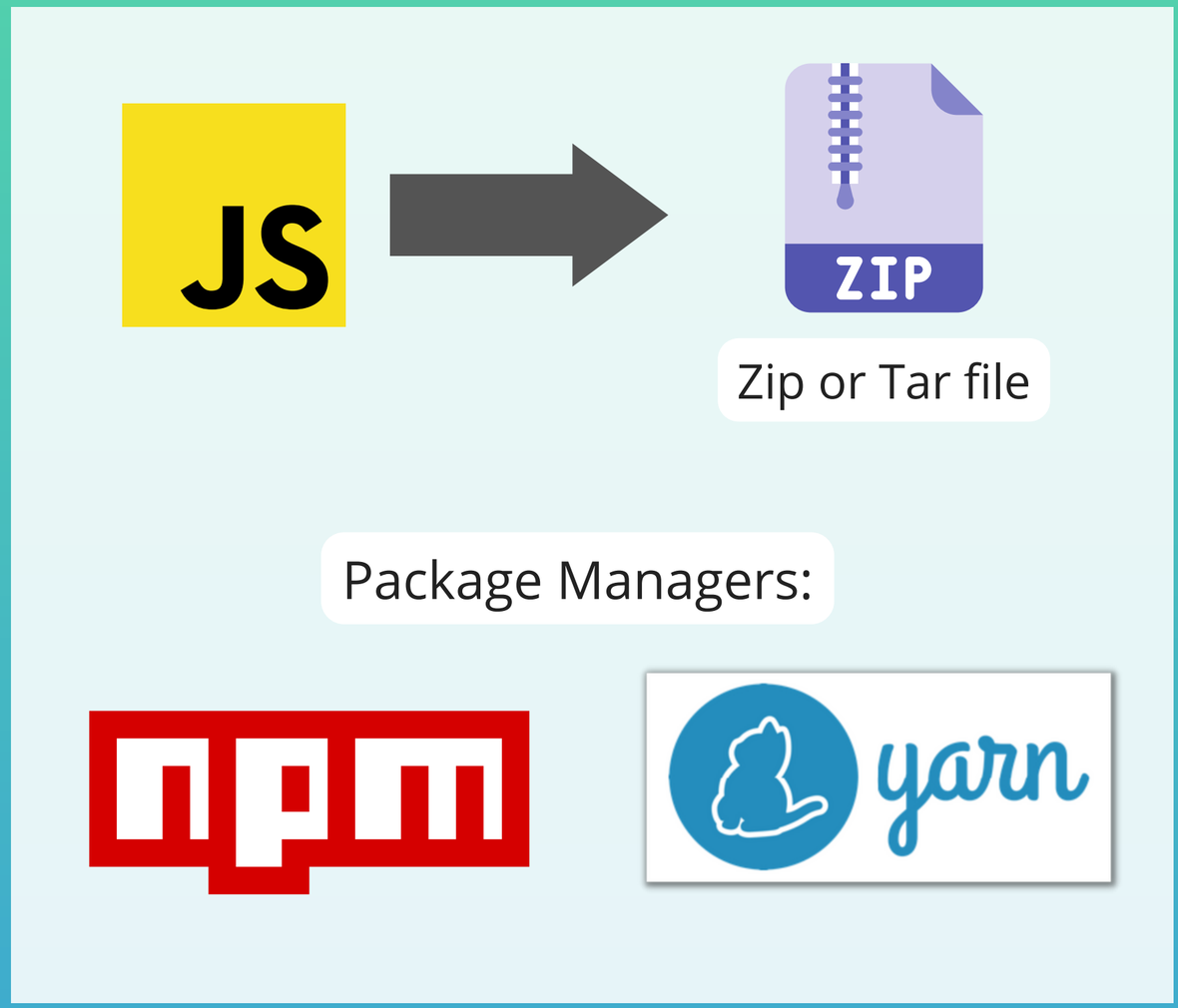
On a **deployment server**:

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

TECHWORLD WITH NANA

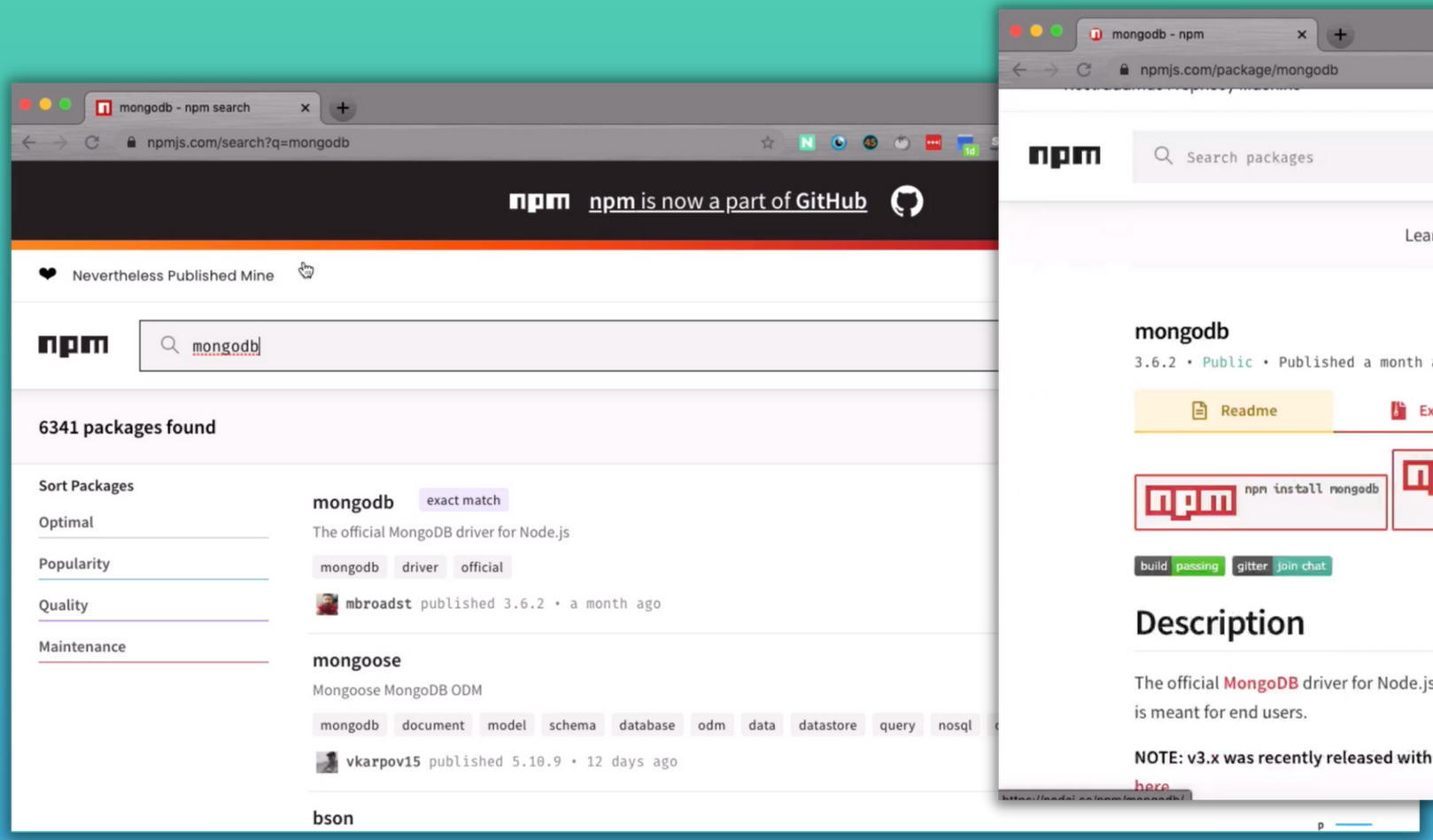
Build JavaScript applications - 1

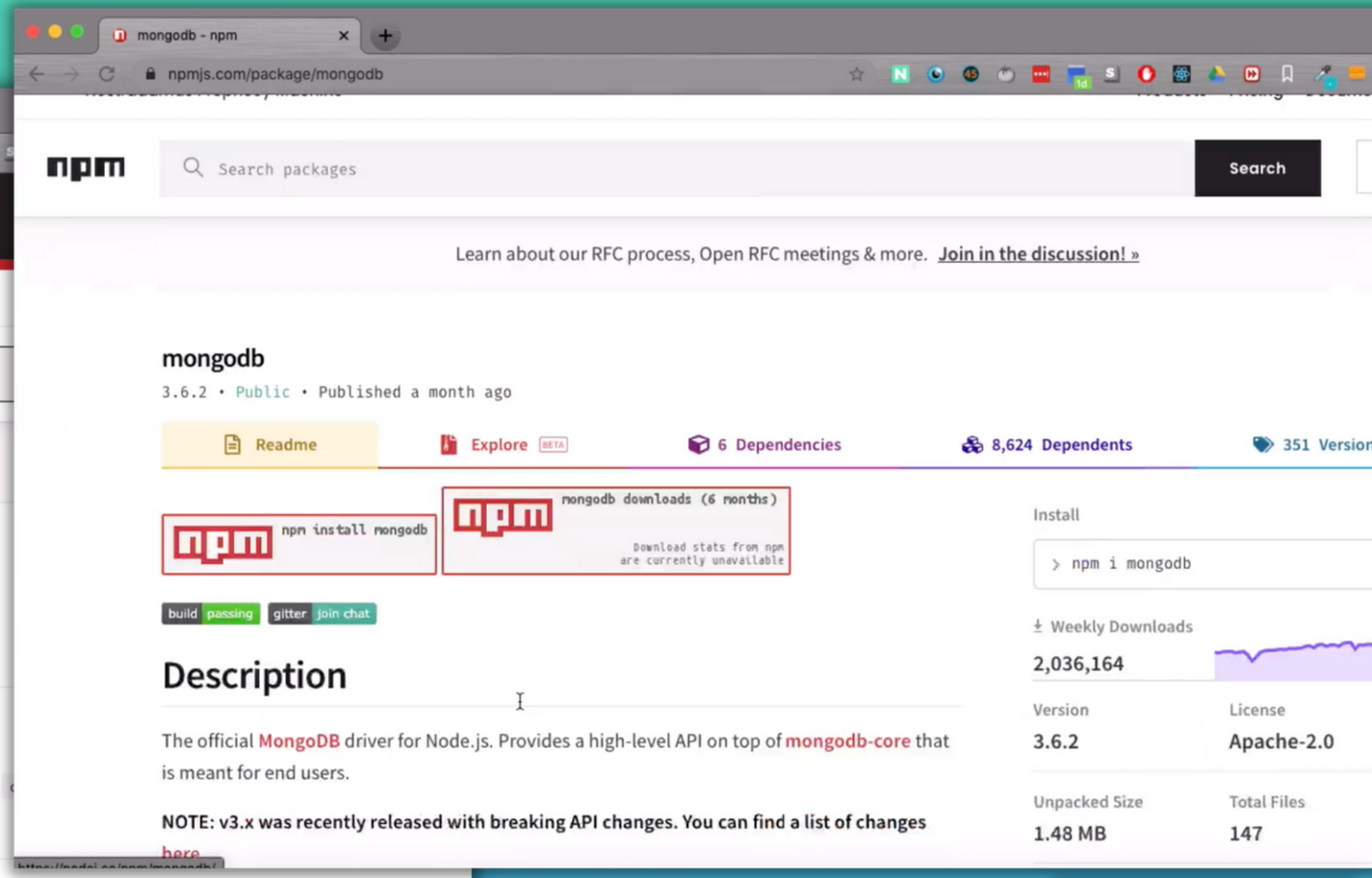
- Artifact of a JavaScript application is e.g. a **zip or tar file**
- But no special artifact type defined
- In JS we have **package managers** and NOT build tools
- For JS we have npm and yarn
- **package.json** file for dependencies
- Package manager install dependencies, but not used for transpiling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



Build JavaScript applications - 2

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





TECHWORLD WITH NANA

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?

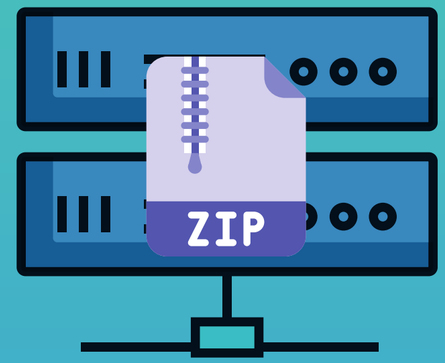
- application code, but **NOT the dependencies**

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

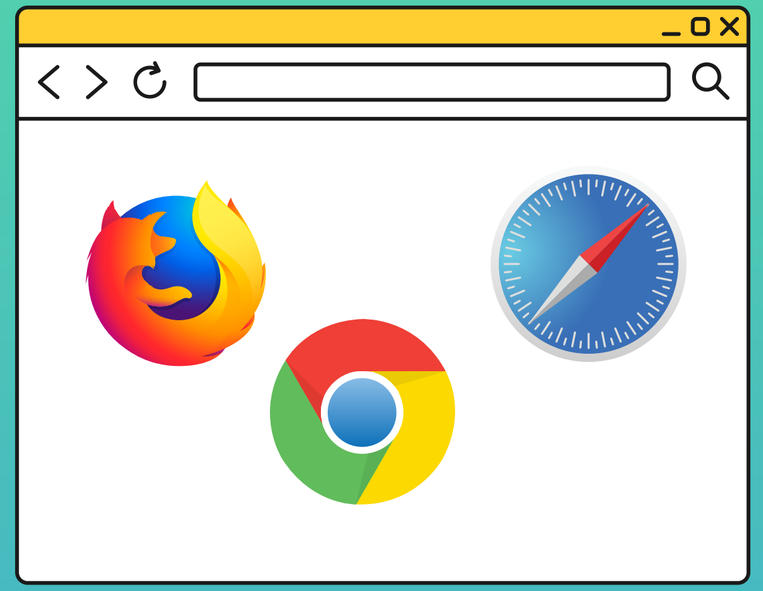
If application consists of JavaScript in Frontend and Backend:

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

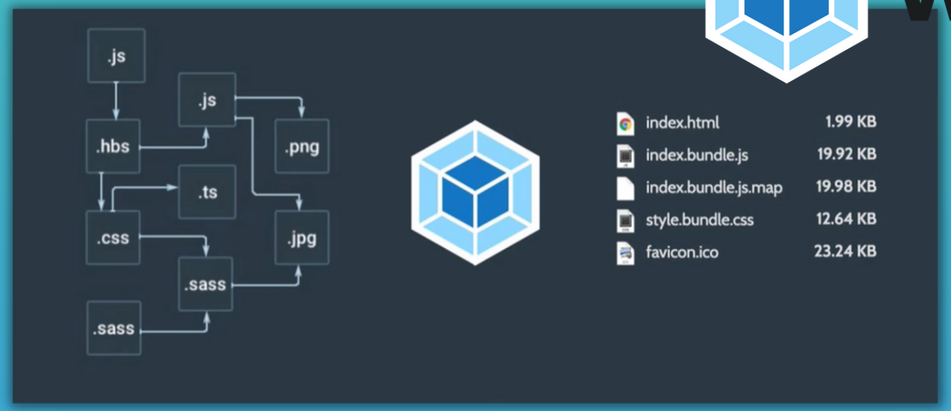
Package Frontend Code - 1

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:**

webpack



- transpiles | minifies | bundles | compresses the code

TECHWORLD WITH NANA

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

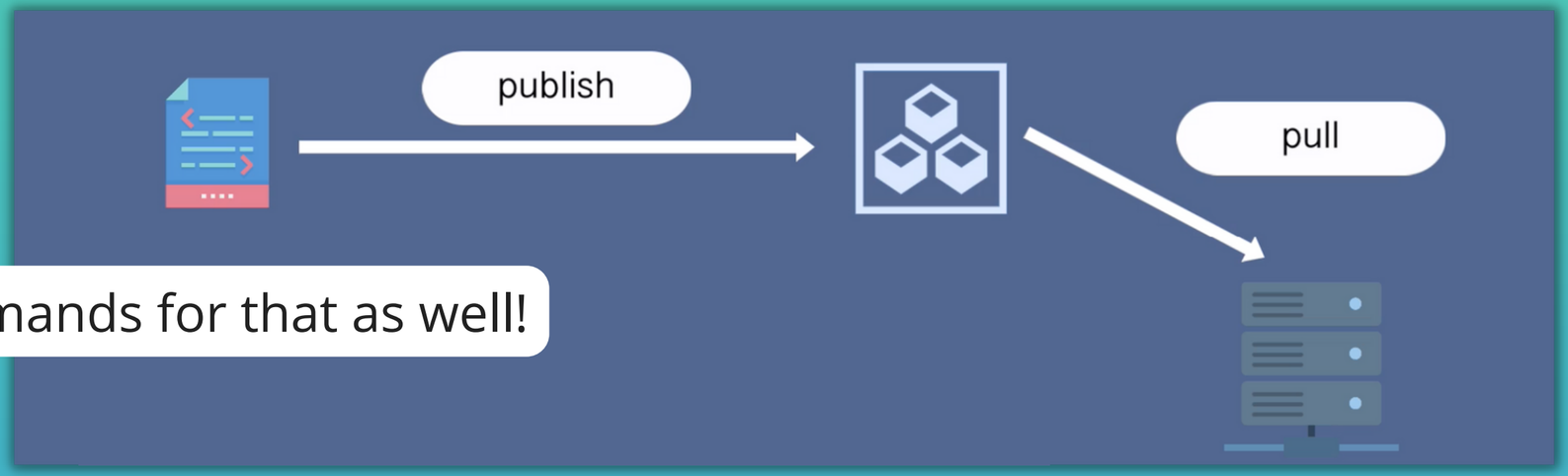
But, concepts very similar to other programming languages

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, pip, dep, maven, yarn

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

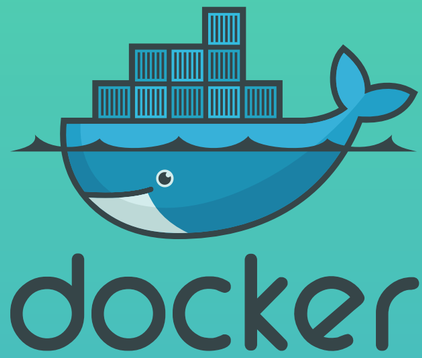


Build Tools have commands for that as well!

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



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

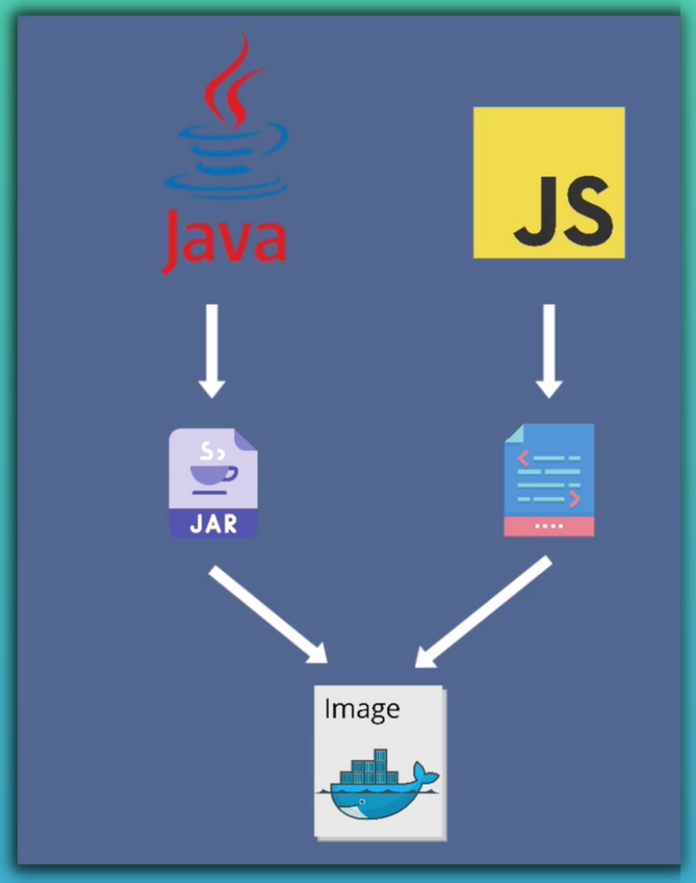
- Docker Image is an alternative for all other artifact types

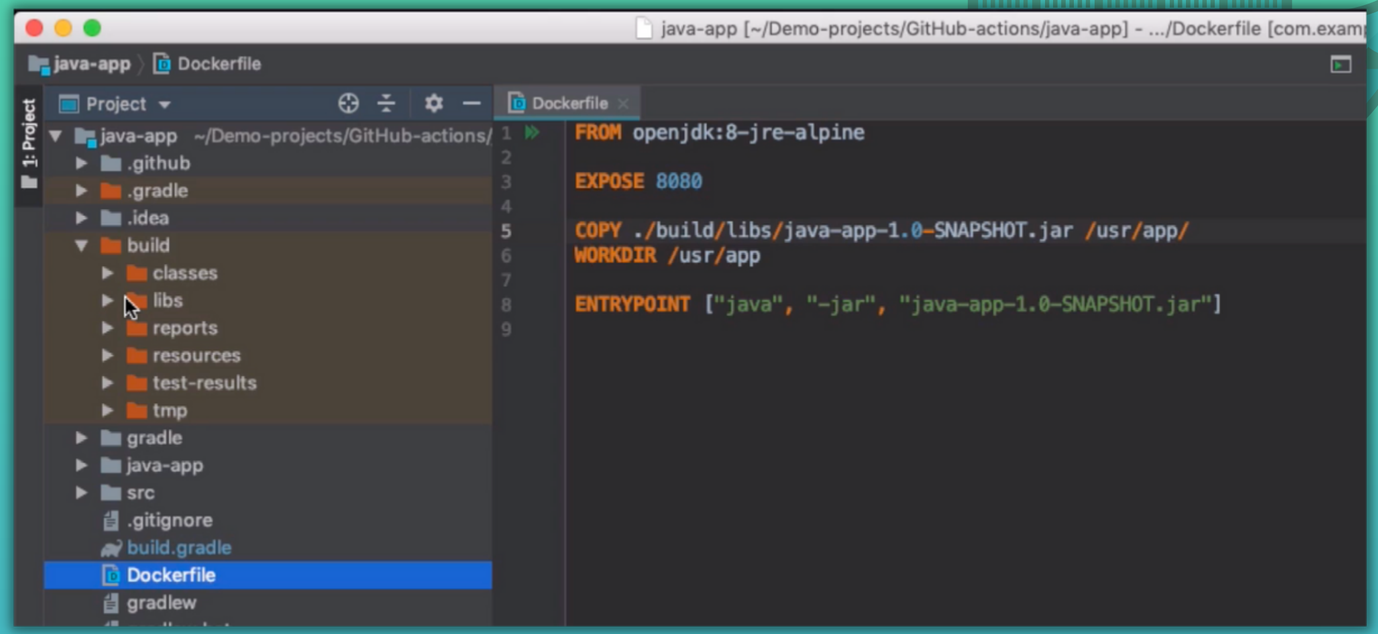
- You don't need to install npm or java on the server

Execute everything in the Image

TECHWORLD WITH NANA

Build Tools and Docker - 3





- You still need to build the application!

More about Docker in later Module!

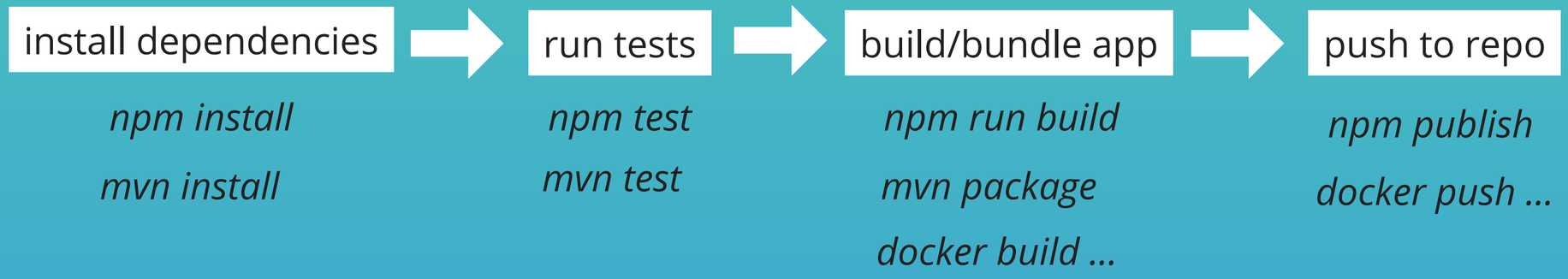
TECHWORLD WITH NANA

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