UNIVERSITY OF MISKOLC,
FACULTY OF MECHANICAL ENGINEERING
AND INFORMATION TECHNOLOGY

Software Technology Lab

GEIAL316-B2a

Continous Integration (CI) tools

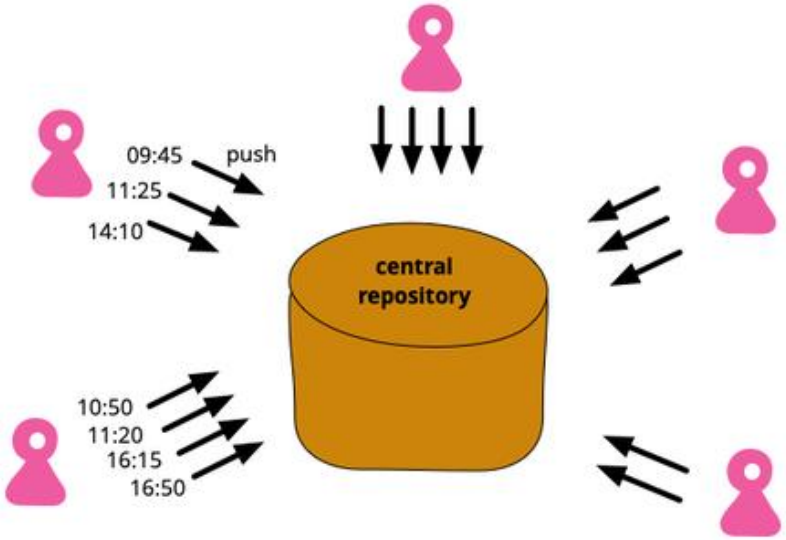
Tamás Tompa, PhD
assistant professor
Department of Information Technology

Miskolc, 2026

Continuous integration

- „Continuous Integration is a **software development practice** where **each member of a team merges their changes into a codebase** together with their colleagues changes **at least daily**. Each of these integrations is **verified by an automated build** (including test) to detect integration errors as quickly as possible."

Martin Fowler



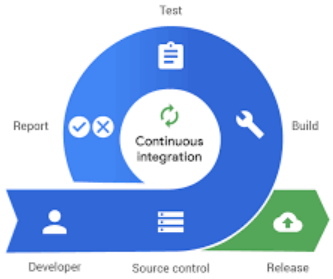
Source: https://martinfowler.com/articles/continuousIntegration.html

Continuous integration

- Modern software development practices that **automate code build, testing, and deployment**
- **Frequently** (several times a day) **code changes are integrated into the main** branch (main/master branch)
- Quick feedback that **the new code didn't mess up the system**
- **related process of Continuous Delivery** (CD)
  - **Automatic installation of** successfully tested code in a test or production environment

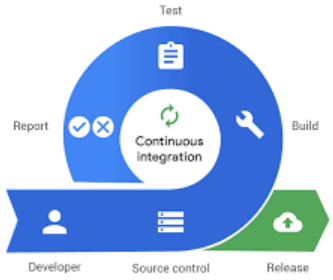
- **CI/CD Tools**
  - Jenkins (Open Source)
  - GitHub Actions (built-in CI/CD on GitHub)
  - GitLab CI/CD (GitLab's own integrated system)
  - CircleCI (cloud-based CI/CD)
  - Travis CI (integrates with GitHub projects)
  - Azure DevOps (Microsoft)

CI processes



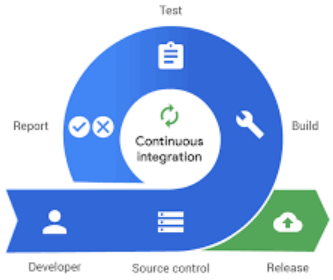
- **Code review**: run automatic code validation to improve code quality
  - syntax and style checkstyle
- **Run unit tests**: run unit tests after code changes
- **Build process**: automate the build process of the software
  - source code compilation
  - create executable files or packages

CI processes



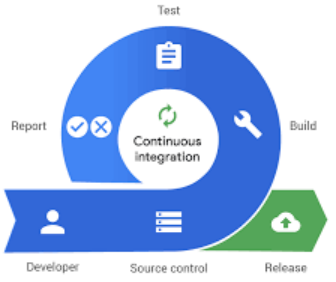
- **Generate documentation**: automatically generate documentation from source code
  - e.g. API documentation, user guides
- **Code Coverage Check**: checking code coverage of unit tests to ensure code quality
  - min. 80%
- **Run functional tests**: automate functional tests for your application to evaluate code changes.

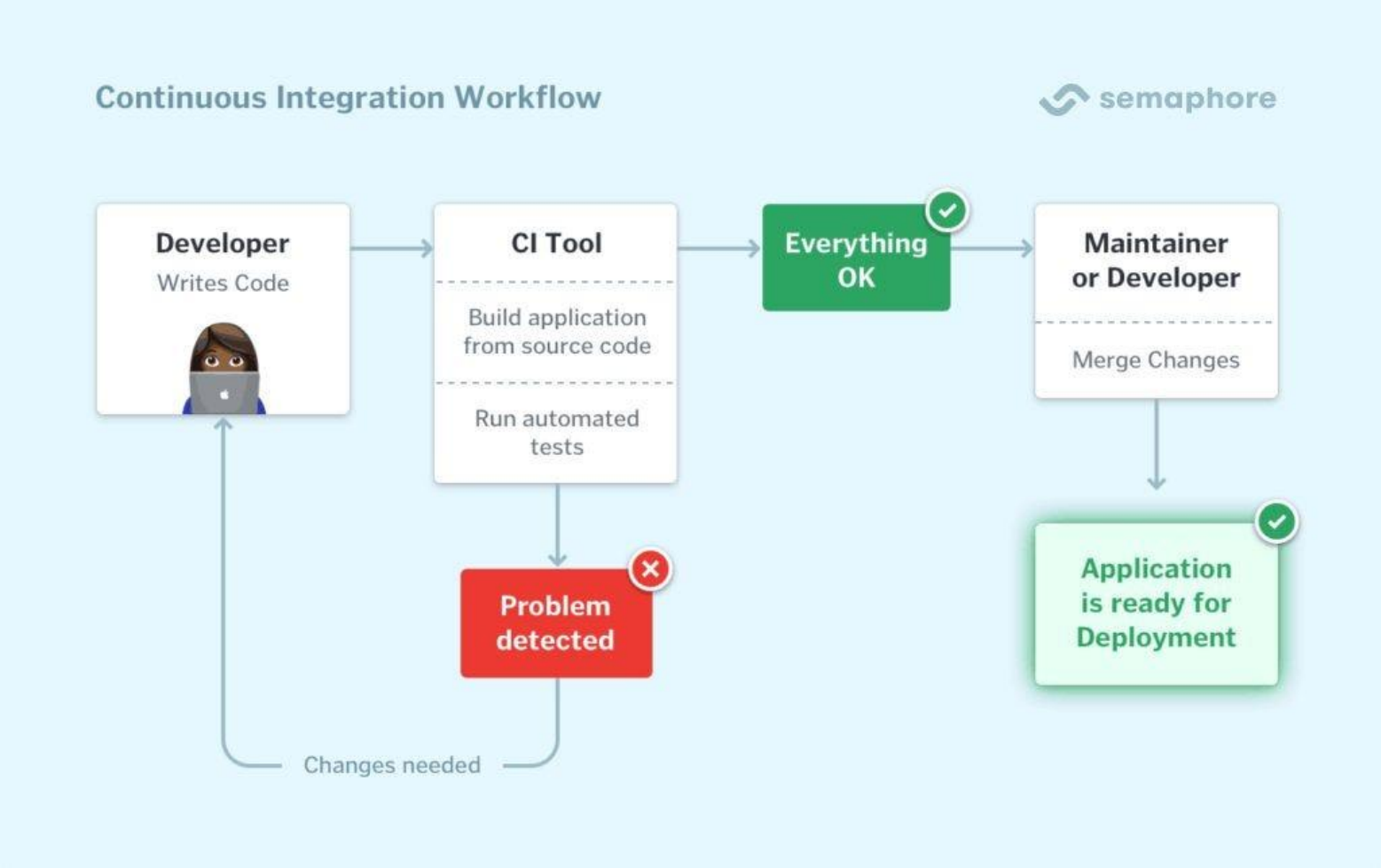
CI processes



- **Run performance tests**: automated testing of app's performance
- **Security checks**: run automatic security checks to identify potential security risks
- **Send notifications**: automatically send notifications about changes in the CI system, such as successful or failed builds, running tests, and more.

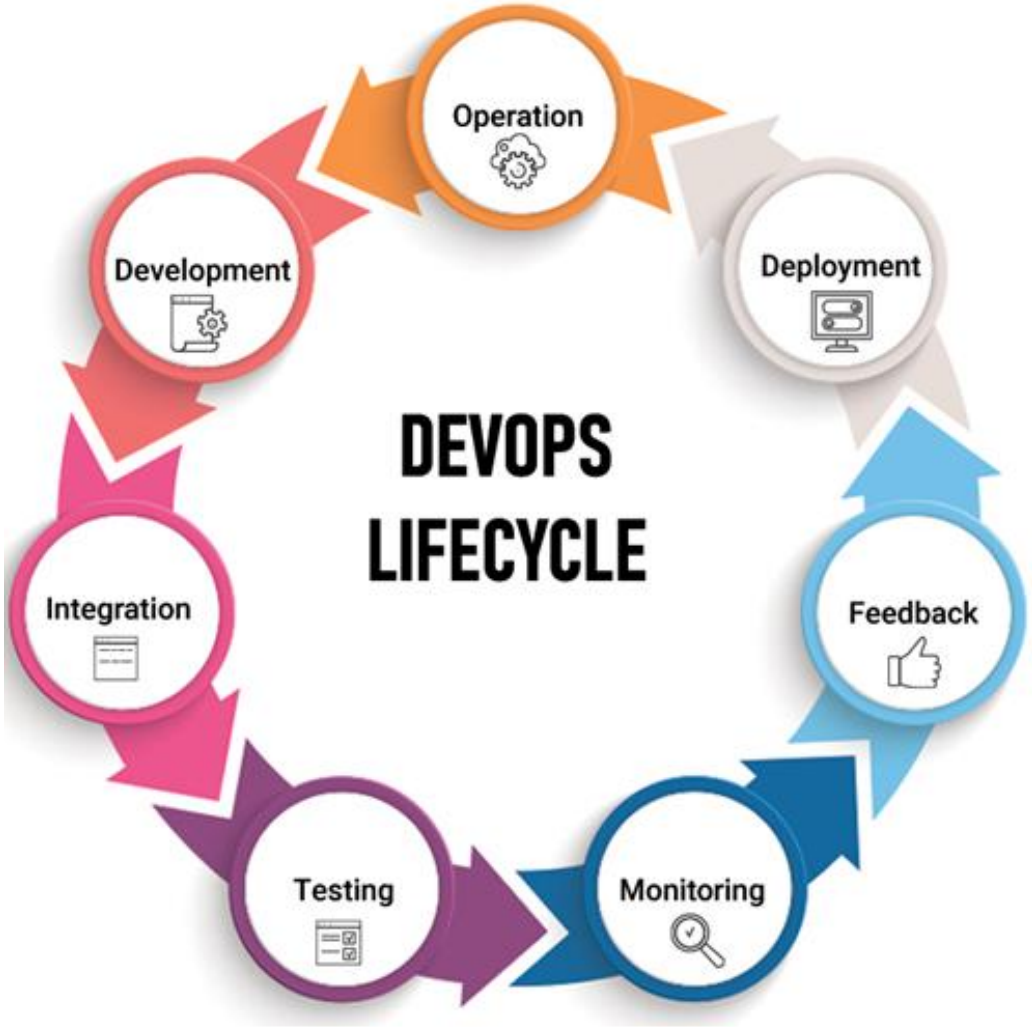
CI processes





Forrás: https://medium.com/@AbbasPlusPlus/the-continuous-integration-flow-a-step-by-step-guide-7e32a08fd54c

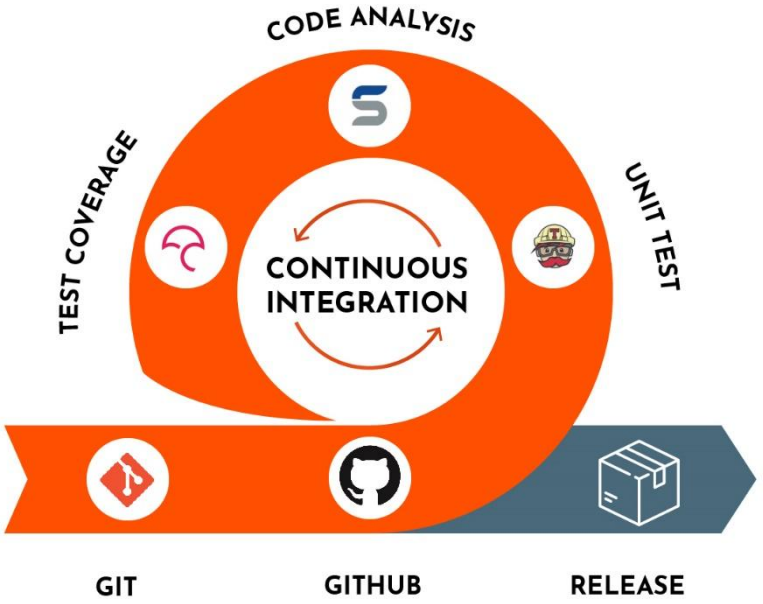
CI Life Cycle



Source: https://www.spiceworks.com/tech/devops/articles/what-is-devops-lifecycle/

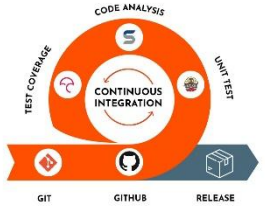
CI Life Cycle - Processes

1. Design
2. Development (coding)
3. Version control
4. Code review
5. Build
6. Unit testing
7. Test Result Evaluation
8. Send Notification
9. Repair
10. Repeat process



Source: https://devonblog.com/wp-content/uploads/2019/07/c-i-100.jpg

CI Life Cycle - Processes

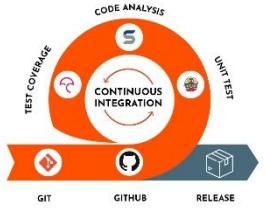


- Developers manage **code in a version control system** (e.g. Git, Mercurial, SVN)
  - new code changes are made in branches (e.g. feature-xyz)
  - integrate the code into the main branch (main/master) using pull request (PR) or merge request (MR)

- In the development environment, the **CI system** (e.g. Jenkins, GitHub Actions, GitLab CI/CD) **detects new changes**
  - compiles the code

- CI system **runs various tests**, if a test fails, **the developer is notified**

CI Life Cycle - Processes



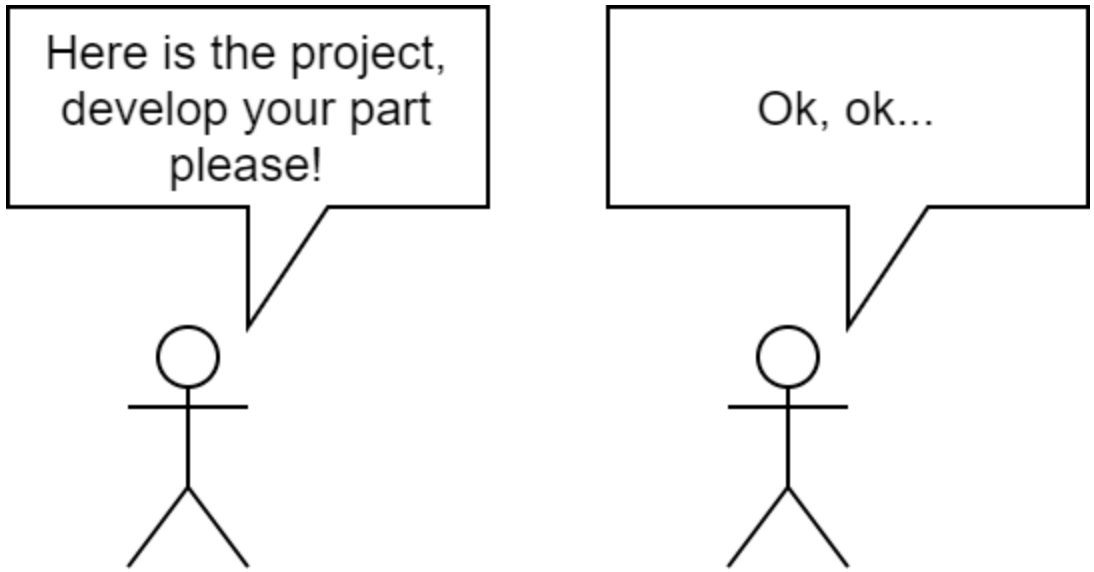
- **Code Quality & Static Analysis**
  - linting: checks code standards (e.g. ESLint, Prettier, Pylint, SonarQube)

- **If the build and tests are successful**, the output **files** (binaries, Docker images, packages) **are stored** in a repository (e.g. Artifactory, Nexus, Docker Registry)

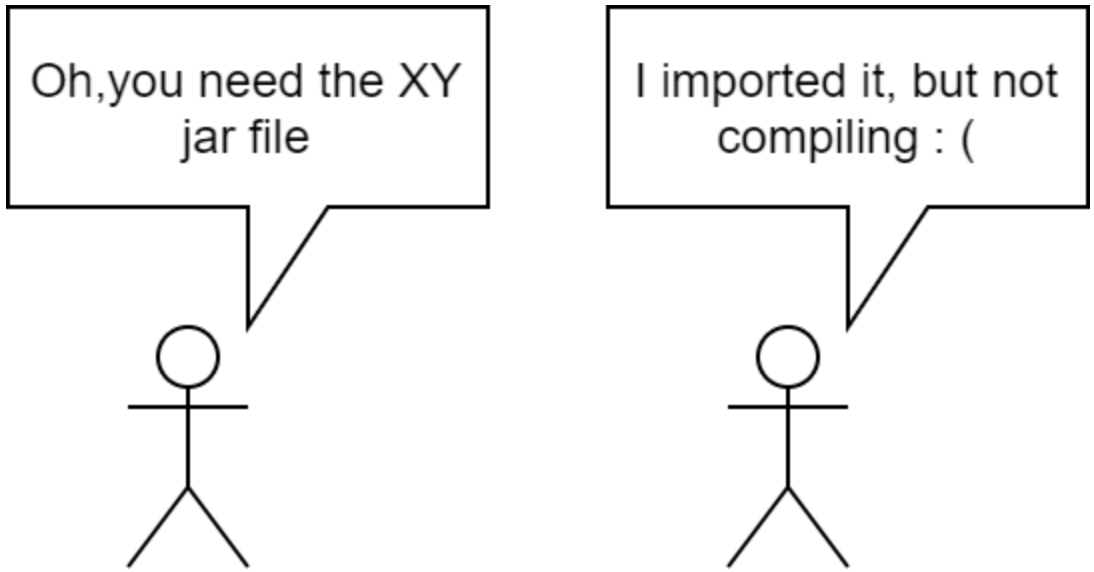
- The CI system notifies developers of the **build and testing results via email through a different channel**

- **If the CI process is successful, the code is passed to the** Continuous Deployment stage

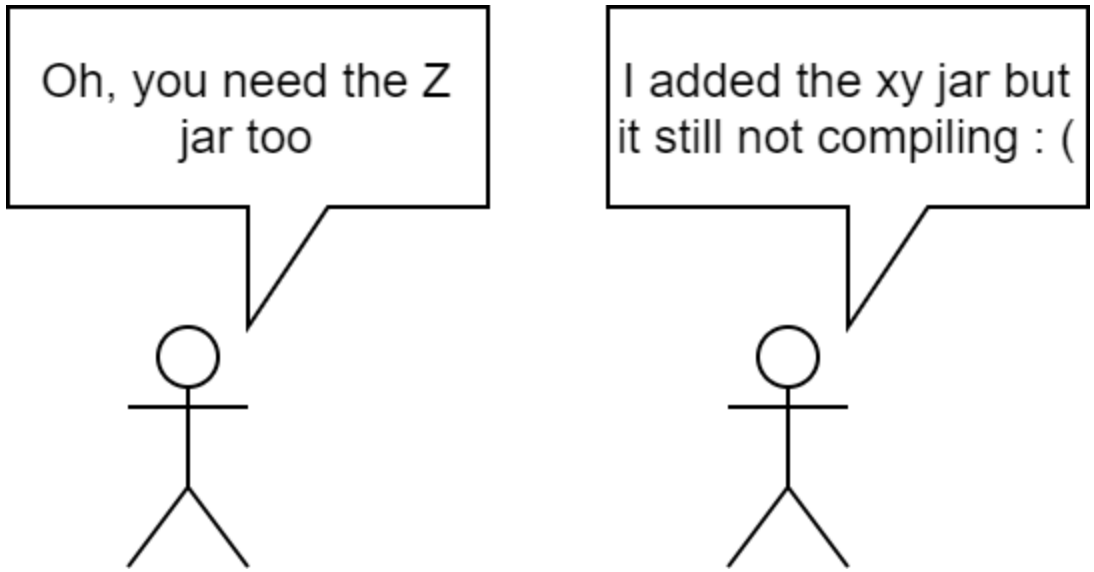
CI build tools - why



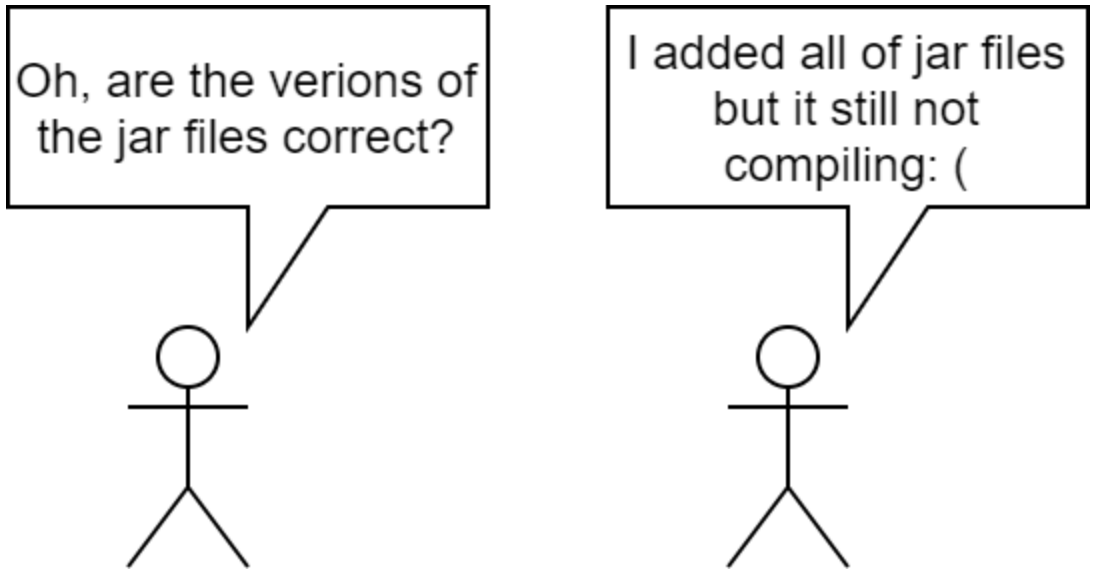
CI build tools - why



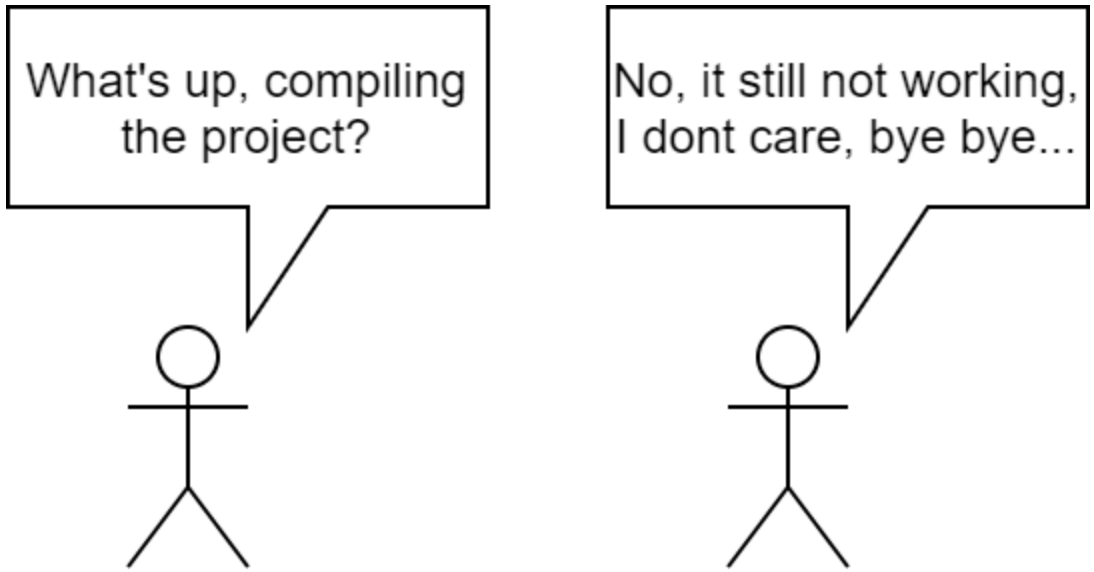
CI build tools - why



CI build tools - why



CI build tools - why



CI build tools

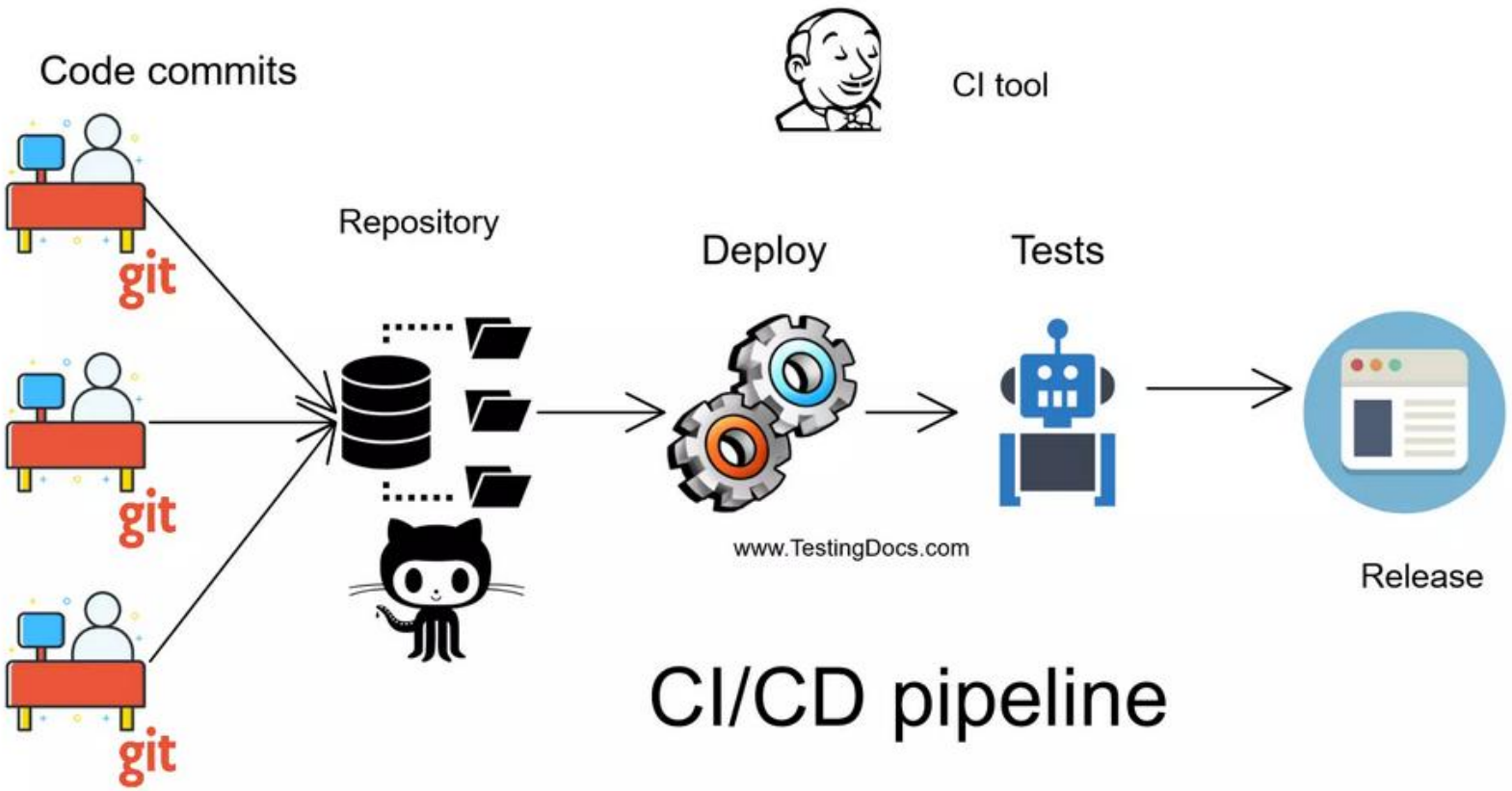
- **„A build tool is a tool that automates everything related to building the software project."**

  Jakob Jenkov

  - Source code generation
  - Generating documentation based on source code
  - Source code translation
  - Compiled code packaging (Jar, Zip, War, etc.)
  - Installing packaged code on a server, in a repo (or elsewhere)

Filed under: https://jenkov.com/tutorials/maven/maven-tutorial.html

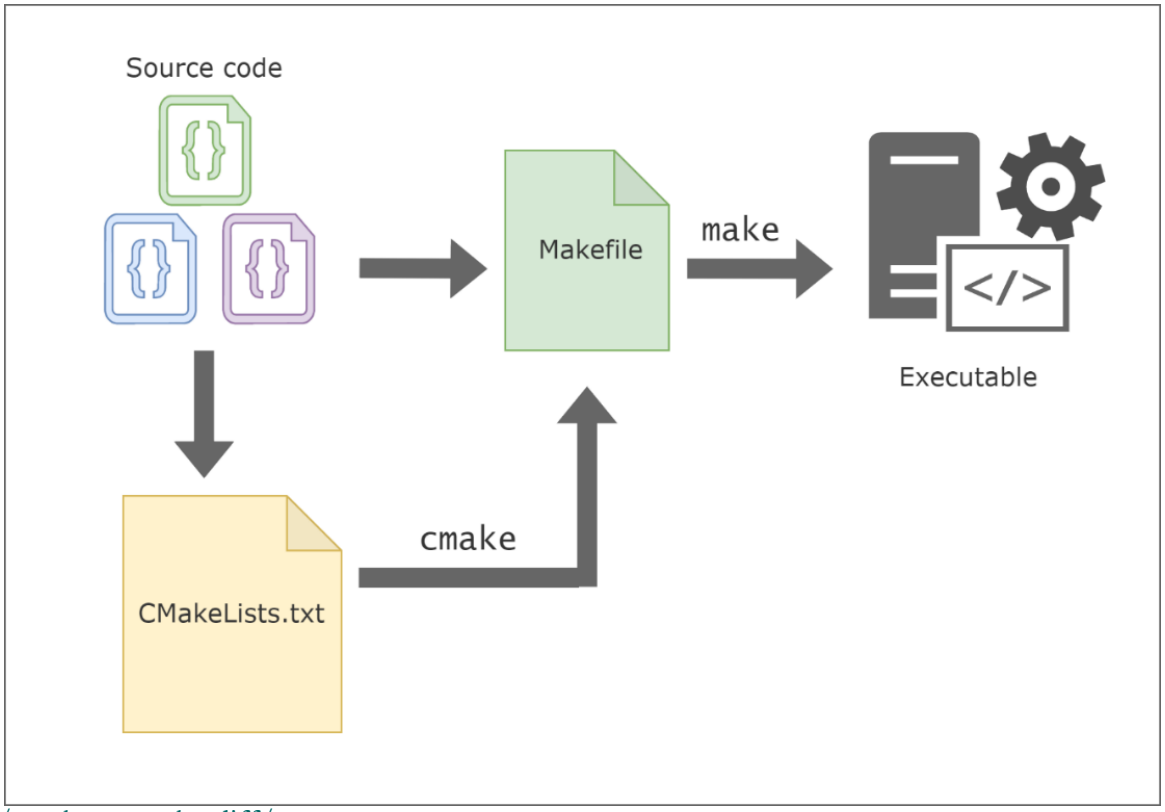
CI build tools



Source: https://www.testingdocs.com/continuous-integration-tools/

CI build tools - Make

- **Make**
  - C/C++
  - automatic build, compile settings



Source: https://earthly.dev/blog/cmake-vs-make-diff/

CI build tools - Make

- **Make example**

hellomake.c	hellofunc.c	hellomake.h

`#include <hellomake.h>`
 	
`int main() {`
`  // call a function in another file`
`  myPrintHelloMake();`
 	
`  return(0);`
`}`
	`#include <stdio.h>`
	`#include <hellomake.h>`
	
	`void myPrintHelloMake(void) {`
	
	`  printf("Hello makefiles!\n");`
	
	`  return;`
	`}`
		`/*`
		`example include file`
		`*/`
		
		`void myPrintHelloMake(void);`

Makefile 1

```
hellomake: hellomake.c hellofunc.c
	gcc -o hellomake hellomake.c hellofunc.c -I.
```


Source: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

CI build tools – Java classpath

- **Java classpath**
  - **a list of locations where the JVM will search for classes and other resources**
  - allowing to embed external directories (JAR files)
  - `javac -cp` command

```
javac -cp C:/.../jardir1/*; C:/.../jardir2/* class_with_main_method
```

```
Javak -cp ".:/home/text/mail.jar:/home/text/servlet.jar;"
Myjawafile. Java
```

  - difficult, long commands
    - → apply build tools (Ant, Maven, etc.)

CI build tools - Apache Ant

- **Apache Ant**
  - Make file for Java
  - XML (translation, packaging, tests)
  - reuse of independent modules

Java-only	Ant

`md build\classes`
`javac`
`  -sourcepath src`
`  -d build\classes`
`  src\oata\HelloWorld.java`
`echo Main-Class: oata.HelloWorld>mf`
`md build\jar`
`jar cfm`
`  build\jar\HelloWorld.jar`
`  mf`
`  -C build\classes`
`  .`
 	
`java -jar build\jar\HelloWorld.jar`
	`<mkdir dir="build/classes"/>`
	`<javac`
	`  srcdir="src"`
	`  destdir="build/classes"/>`
	`<!-- automatically detected -->`
	`<!-- obsolete; done via manifest tag -->`
	`<mkdir dir="build/jar"/>`
	`<jar`
	`  destfile="build/jar/HelloWorld.jar"`
	
	`  basedir="build/classes">`
	`  <manifest>`
	`  <attribute name="Main-Class" value="oata.HelloWorld"/>`
	`  </manifest>`
	`</jar>`
	`<java jar="build/jar/HelloWorld.jar" fork="true"/>`

Cover: https://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html
https://en.wikipedia.org/wiki/Apache_Ant

CI build tools - Apache Maven

- **Apache Maven**

  - „Maven is a build tool, a project management tool, an abstract container for running build tasks.”
  - 2004.07.13. - first version
  - project management tool for automating Java-based projects and managing the build process

  - Objectives
    - simplify project management (pom.xml)
    - centralized dependency management (Maven repo: https://mvnrepository.com/)
    - modularity and customizability (plugins)
    - community support (open source, Apache License v2)
    - a unified system for building software projects

CI build tools - Apache Maven

- **Installation**
  - JDK required
  - https://maven.apache.org/download.cgi
  - after unpacking: `C:\apache-maven-3.9.5`
  - set an environment variable
    - to add path: `C:\apache-maven-3.9.5`
      - or `export PATH=/opt/apache-maven-3.9.5/bin:$PATH`
    - then in cmd: `mvn --version` command

```
C:\Users\Tompa_Tamas>mvn --version
Apache Maven 3.9.5 (57804ffe001d7215b5e7bcb531cf83df38f93546)
Maven home: C:\apache-maven-3.9.5
Java version: 18.0.1.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-18.0.1.1
Default locale: hu_HU, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

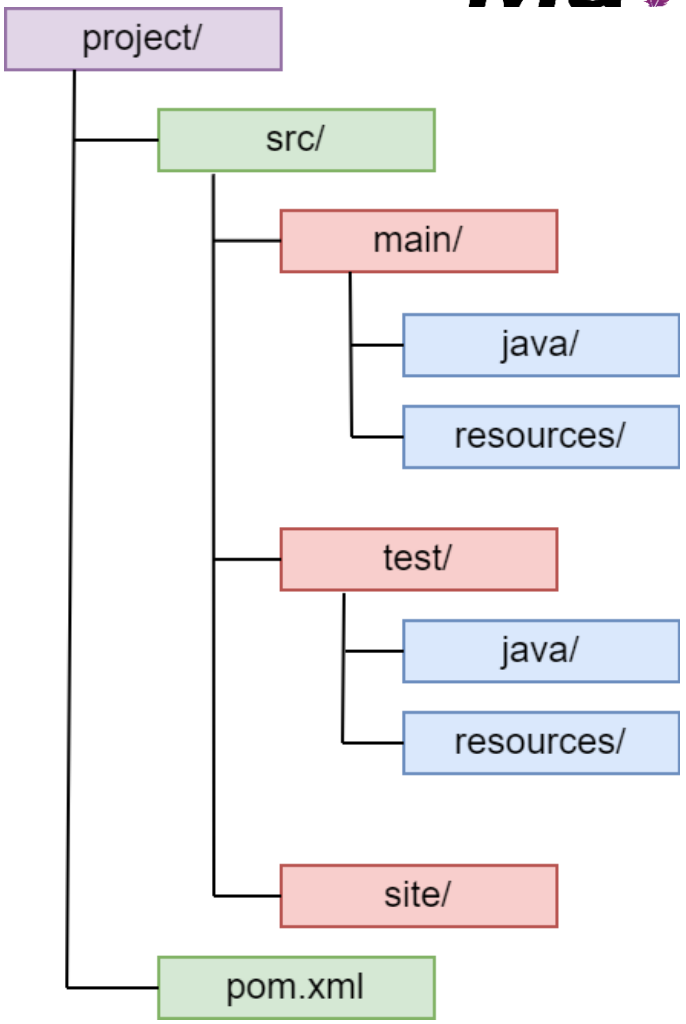
CI build tools - Apache Maven

- **IDE integration**
  - IntelliJ IDEA 
    - Built-in support
  - Eclipse 
    - https://www.vogella.com/tutorials/EclipseMaven/article.html
    - Indigo version upwards already includes
  - Visual Studio Code 
    - Maven for Java plugin
    - https://code.visualstudio.com/docs/java/java-build

- **Official documentation**
  - https://maven.apache.org/guides/index.html

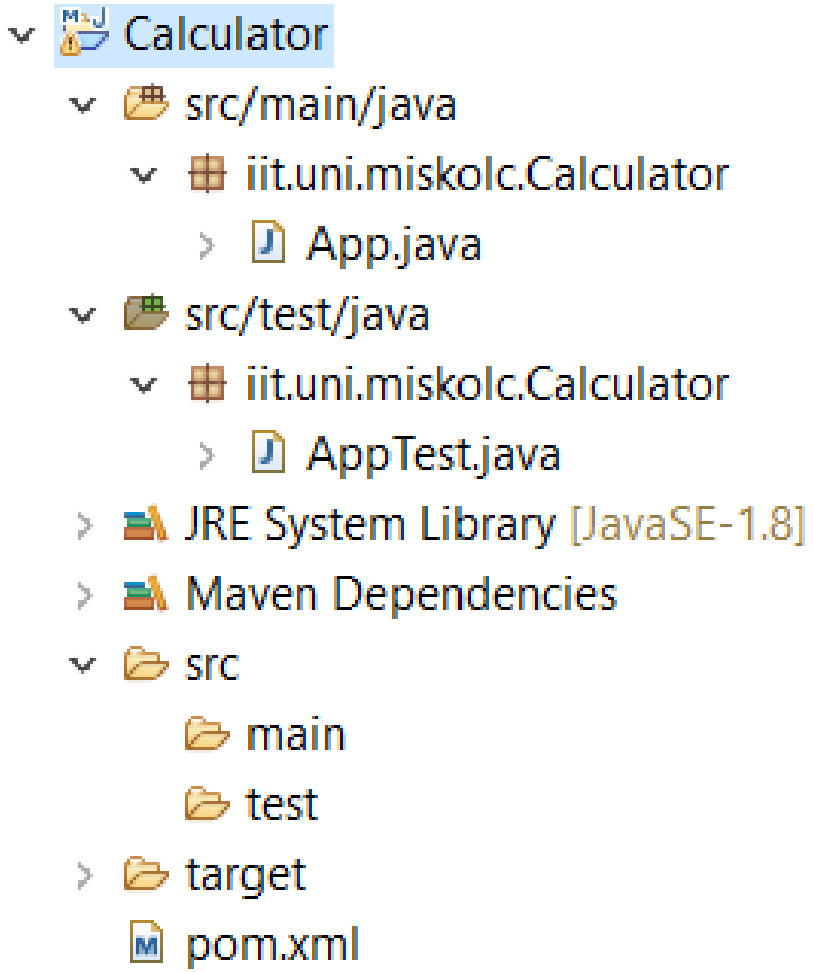
CI build tools - Apache Maven

- **Library structure**
  - standard, fixed structure
  - https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
  - **src/main/java**: source code
  - **src/main/resources**: configuration files, database configurations, language files, etc.
  - **src/test/java**: test source codefor
  - **src/test/resources**: test data, configuration files, mock databases, etc.
  - **pom.xml**: project Configuration File
  - **target**: output generated by the build process



CI build tools - Apache Maven

- **Library structure example**



CI build tools - Apache Maven

- **pom.xml**
  - **Project Object Model**: configurationfile describing the project
  - **Archetpye**: create from a sample (e.g. quickstart)
    - directory structure, framework created by: `mvn archetype:generate`

```
<project>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.mycompany.app</groupId>
      <artifactId>my-app</artifactId>
      <version>1</version>
</project>

<dependencies>
       <dependency>
            …
       </dependency>
</dependencies>
```

CI build tools - Apache Maven

- **pom.xml**

  - `<project>`: the root element of the project, which contains all the additional elements

  - Maven koordináták: `groupid:artifactid:version`
    - `<groupid>`: unique group identifier for the project
      - `e.g.: org.apache.maven`
    - `<artifactid>`: a projekt neve
      - `pl.:my-project`
    - `<version>`: the version number of the project
      - `pl.: 3.4.5; 1.0-SNAPSHOT`
      - `snapshot:` under active development, a momentary (internal) state of the project
      - `Release:` Releaseable, Stable version

CI build tools - Apache Maven

- **pom.xml**

  - `dependencies:` an element containing the project's dependencies

```
<dependencies>
        <dependency>
            …
        </dependency>
</dependencies>
```

  - `<build>:` the element that contains the build configuration
  - `<plugins>:` an item that contains the configuration of Maven plugins
  - `<repositories>:` an item containing the Maven repository configuration

CI build tools - Apache Maven

- **pom.xml example**


```xml
<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>hu.iit.unimiskolc</groupId>
    <artifactId>Calculator</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>Calculator</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
        <dependency>
```

CI build tools - Apache Maven

- **settings.xml**
  - 2 optional `settings.xml` files:
  - Maven install folder: `$M2_HOME/conf/settings.xml`
  - User home folder: `${user.home}/.m2/settings.xml`
  - Specifying the route of a local repo
  - Specify an active build profile
  - https://maven.apache.org/settings.html

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <interactiveMode/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

CI build tools - Apache Maven

- **Modules**
  - divide the project into parts
    - separate compilation units
    - can be compiled, tested and packaged independently
    - each module has its own pom.xml
    - e.g.: decomposition of business logic and GUI
  - a module can be a library or a functional unit
  - individual modules can refer to other modules as dependencies
    - can be specified in the `pom.xml`

CI build tools - Apache Maven

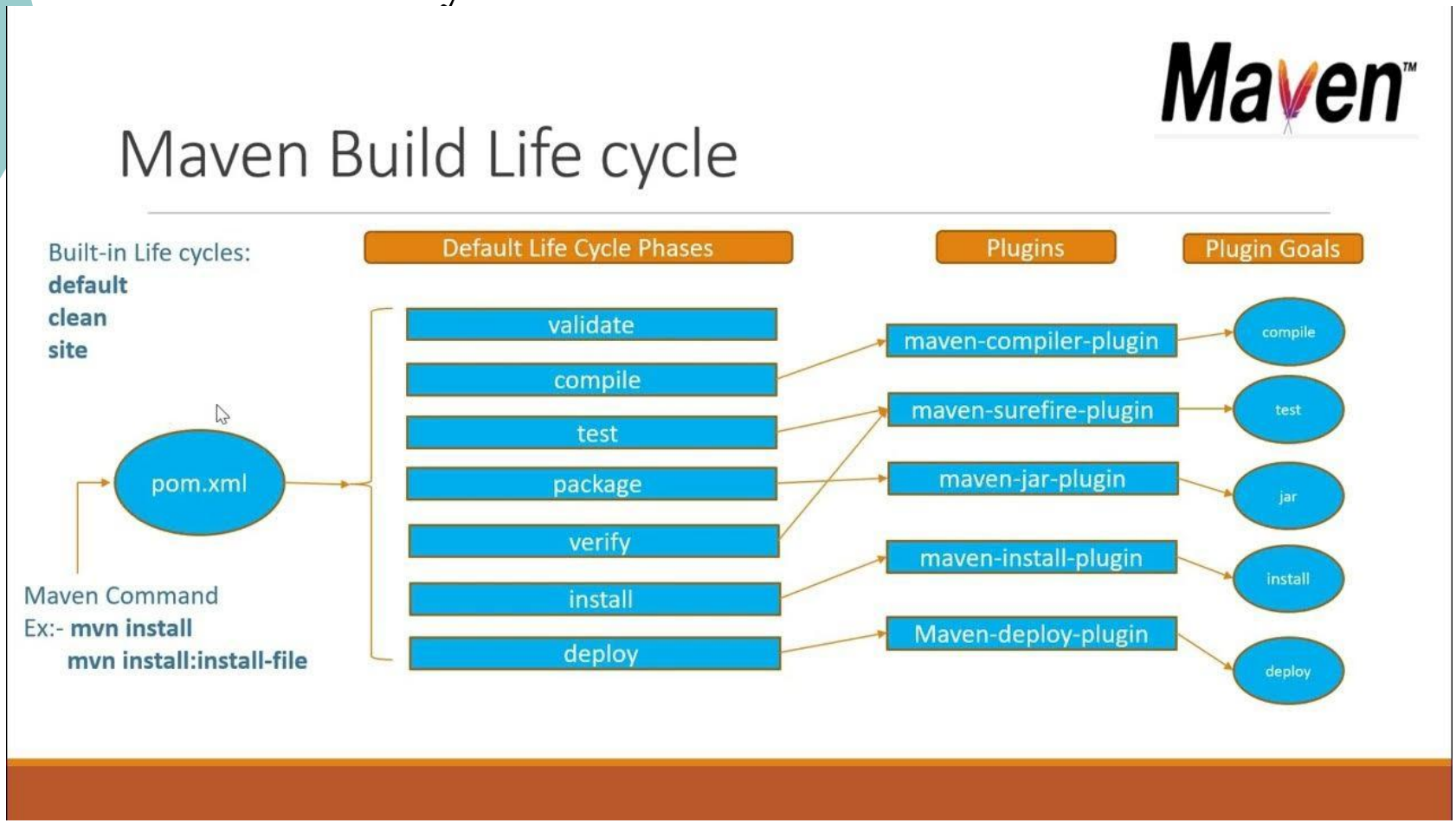
- **Modules example**

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>parent-project</artifactId>
    <version>1.0.0</version>
    <packaging>pom</packaging>

    <modules>
        <module>business-logic</module>
        <module>web-ui</module>
    </modules>
</project>
```

- parent project: `parent-project`
- modules: `Business-Logic` and `Web-UI`

CI build tools - Apache Maven

- **Maven lifecycle**



Source: https://www.youtube.com/watch?app=desktop&v=Yw5KnoH0MSo

CI build tools - Apache Maven

- **Maven lifecycle**

  - **Clean:** delete files created in the previous build. This includes deleting the target folder and build files
  - **Validate:** validate the structure and configuration of the project
  - **Compile:** compile source codes according to the specified language standard, create a target folder
  - **Test**: run the test cases defined in the project
  - **Package**: packaging of compiled source code and other required resources in a specific format (e.g. JAR, WAR, etc.).

CI build tools - Apache Maven

- **Maven lifecycle**

  - **Verify:** checking files created during packaging to ensure that they comply with predefined quality guidelines and rules
  - **Install:** the packaged files to the local Maven repository (.m2). These files are then available to other projects to use as dependencies
  - **Deploy:** publish the packaged files to other remote repositories

CI build tools - Apache Maven

- **Maven Lifecycle Commands**

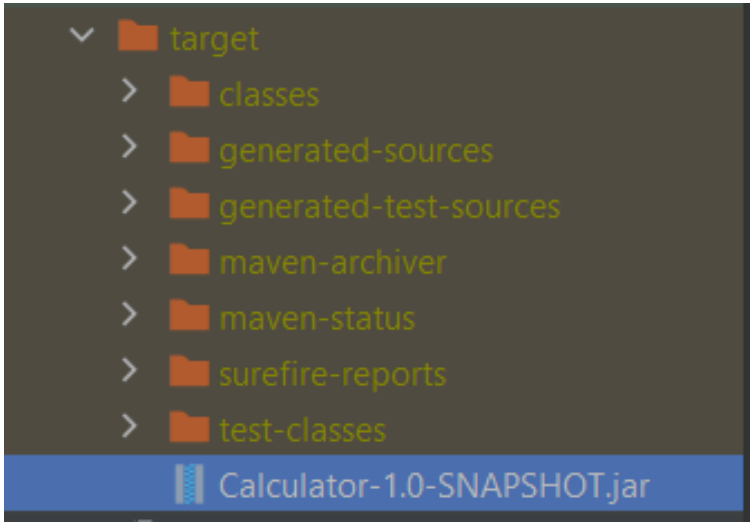
  - `MVN Clean`: delete the target folder (previous build files)
  - `mvn compile`: compile source code in src/main/java
  - `mvn test`: runs tests in the src/test/java folder
  - `mvn package`: package compiled source code into an executable file (pl. JAR)
  - `mvn install`: install the package in your local Maven repository (.m2)
  - `mvn deploy`: publish the package to a specific remote repository

  - https://jenkov.com/tutorials/maven/maven-commands.html

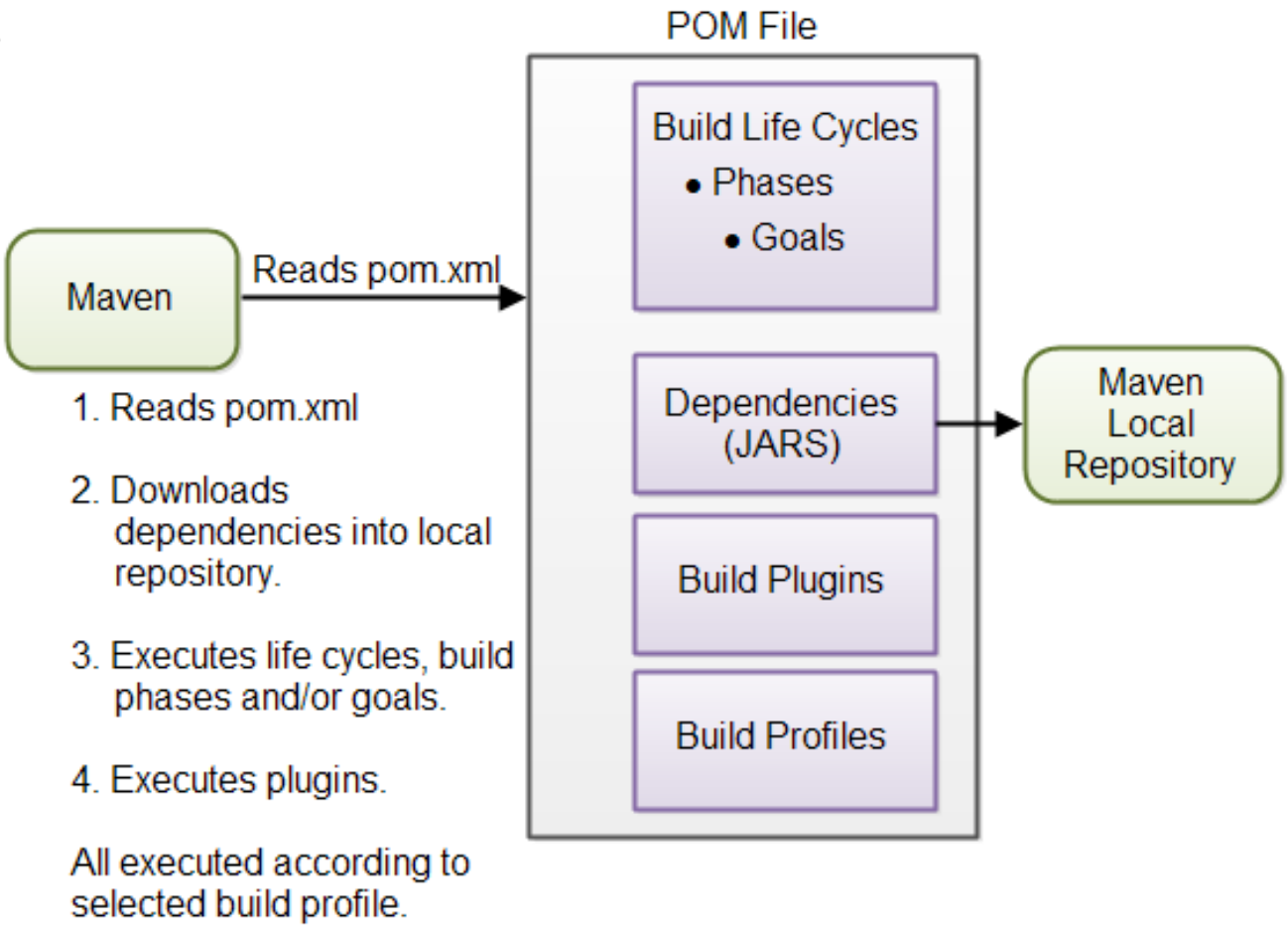
CI build tools - Apache Maven

- **Barrel JAR**
  - created by Maven
  - all jar files containing dependencies
  - `pom.xml` n configurable
    - `<build>` → `<plugins>` → `<configuration>`

  - Created in the target folder :



CI build tools - Apache Maven



Source: https://jenkov.com/images/maven/maven-overview-1.png

CI build tools - Apache Maven

- **Maven summary**
  - It provides 3 main things
    - **Project structure, configuration**
      - fixed directory structure
    - **Addiction Management**
      - pom.xml `<dependency>`
    - **Build and test automation**
      - Maven Lifecycle

CI Tools - Docker

- **A containerization platform** that allows **applications and their dependencies to be packaged**, transported, and run in containers
- It's like having a mini **virtual machine** in which the application runs, but **regardless** of **what the operating system is**

- **Once it works, it works everywhere**
  - ensures that the application works the same way on the development machine, test environment, and production server

- **Containerization → Easier deployment and scaling**
  - an application and all of its dependencies are placed in a single container, making it easy to move to other machines or servers

CI Tools - Docker

- **Fast and resource-efficient**
  - containers are smaller and faster than traditional VMs because they don't require a separate operating system for each instance
- **Support for microservices**
  - multiple small services can run in separate containers
  - e.g. a container for the web application, a container for the database

CI Tools - Docker

- **Basic concepts**
  - **Docker Image**
    - an application and its environment in a pre-made package
  - **Docker Container**
    - a running instance of a Docker Image
  - **Docker file**
    - s file that defines how to create a Docker Image
  - **Docker Hub**
    - a public repository from which you can download pre-built Docker Images

CI Tools - Docker

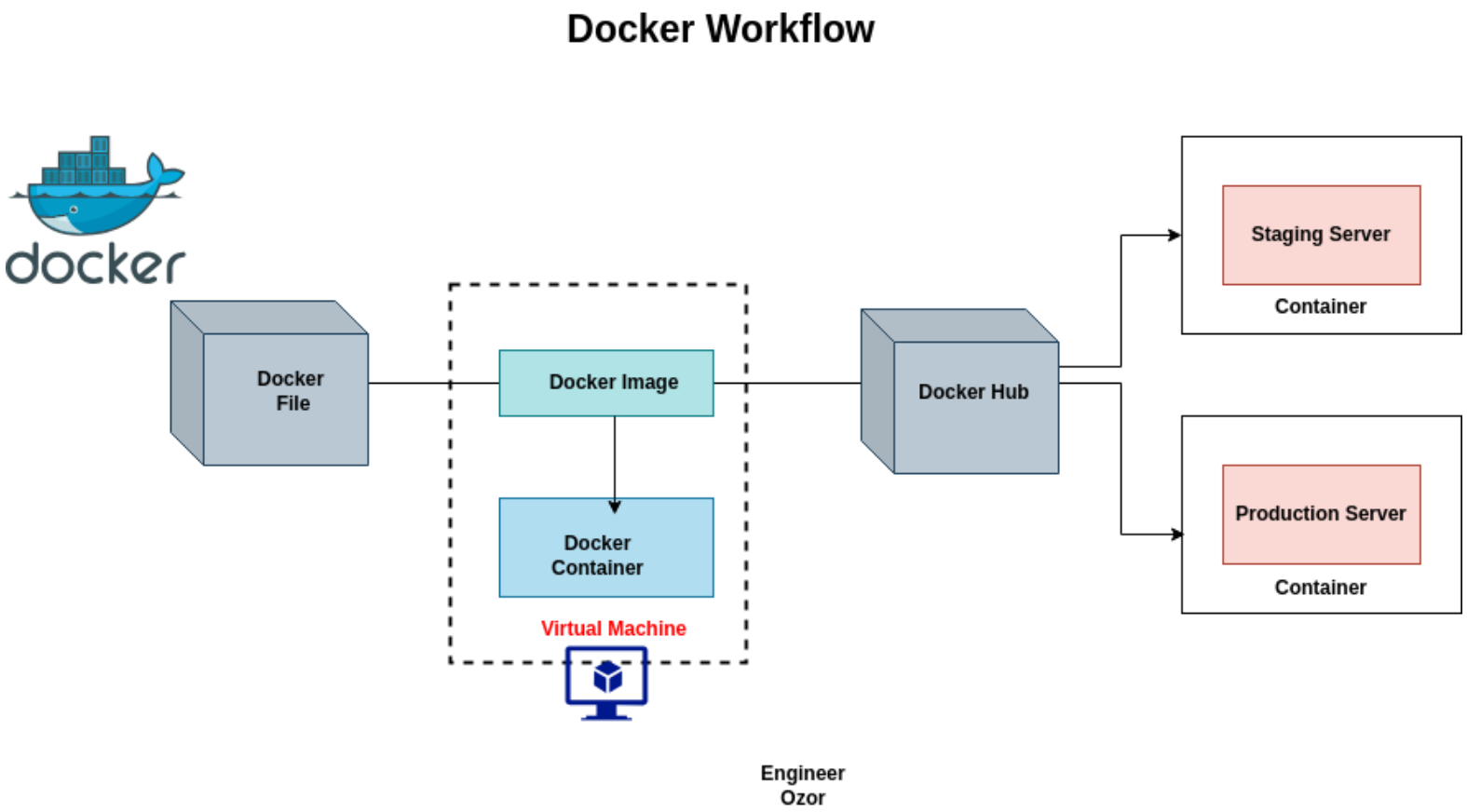
- **Example (node.js)**

```
# Alap image (Node.js)
FROM node:18

# Munkakönyvtár beállítása
WORKDIR /app

# Csomagok bemásolása és telepítése
COPY package.json ./
RUN npm install

# Az alkalmazás fájljainak bemásolása
COPY . .

# Az alkalmazás futtatása
CMD ["node", "app.js"]

# Port megnyitása
EXPOSE 3000
```

```
docker build -t my-node-app .
```

```
docker run -p 3000:3000 my-node-app
```

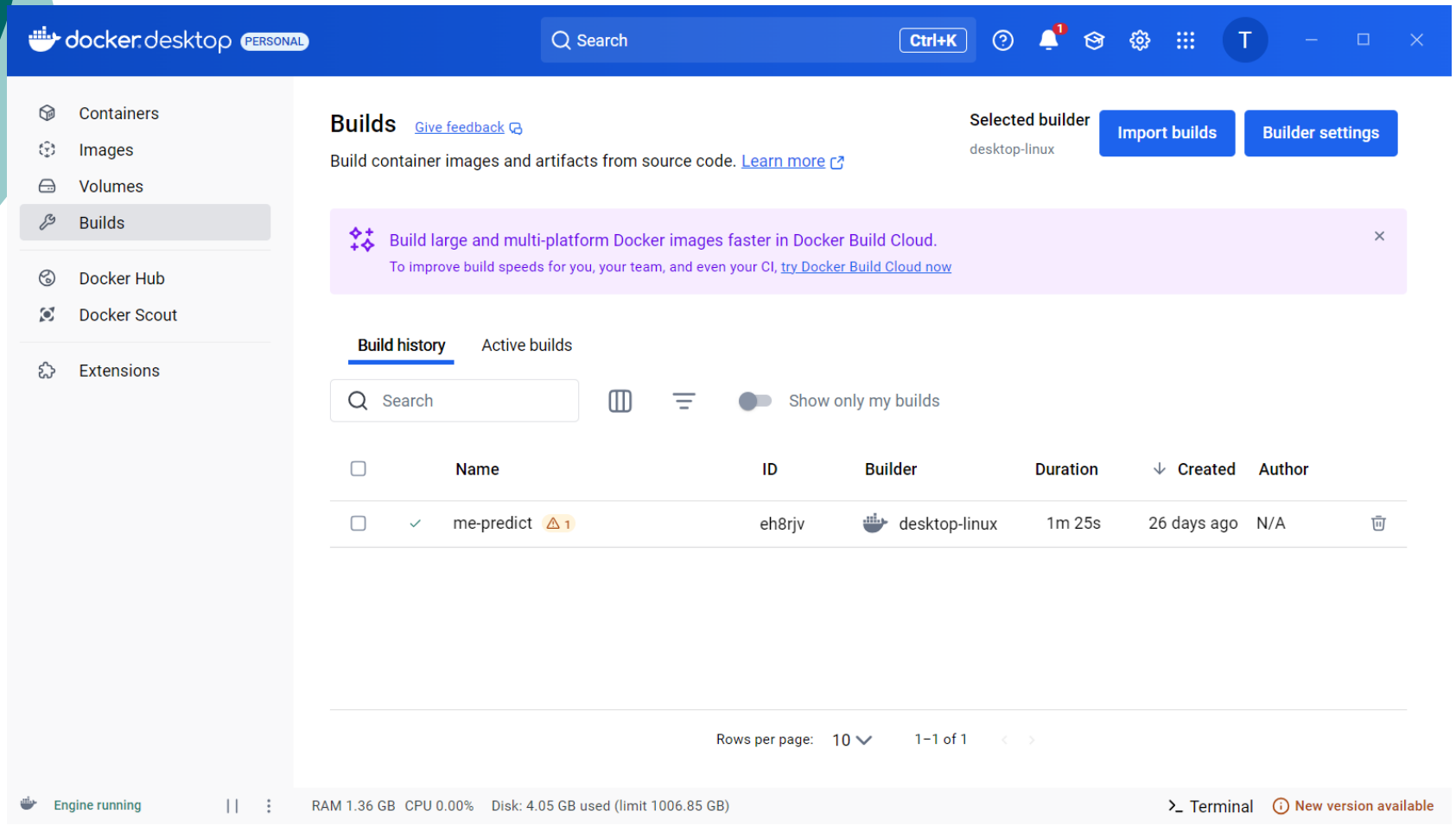
CI Tools - Docker



Source: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184

CI Tools - Docker

- **Docker Desktop – Windows tool**



CI Tools - JIRA

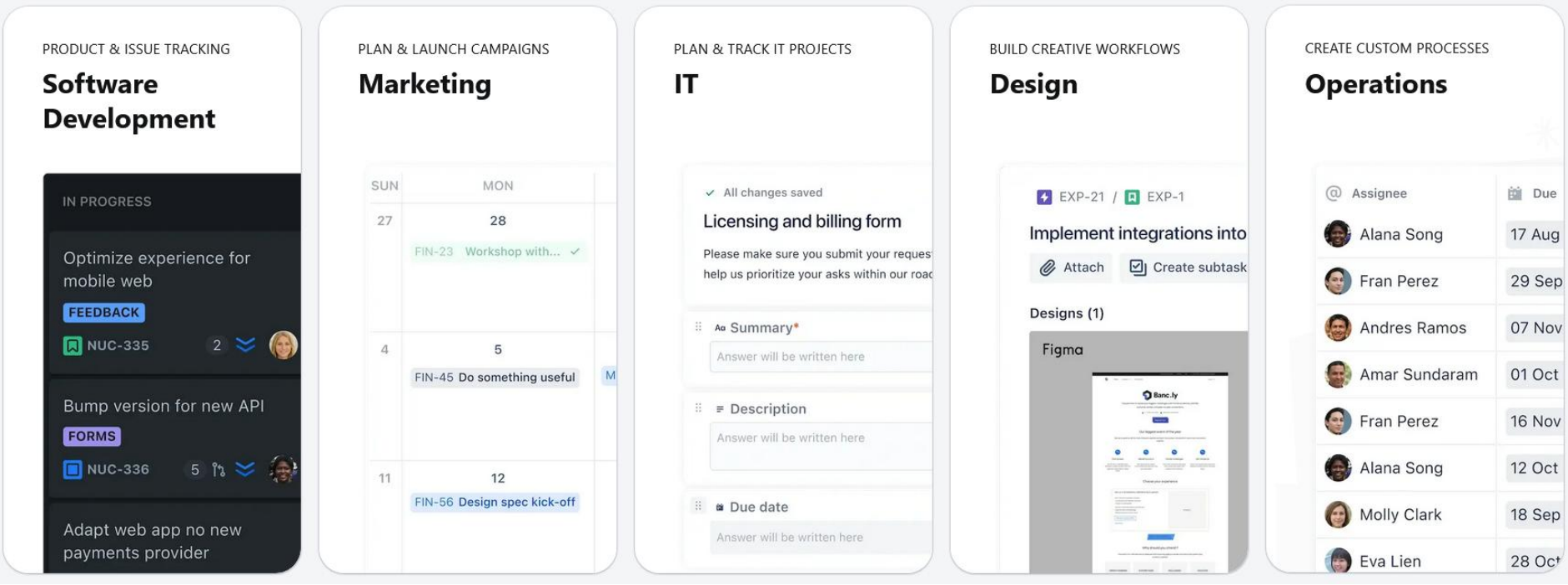
- **Project management tool** developed by Atlassian
- Designed primarily for agile teams to **help plan**, track, and **manage workflows**
- https://www.atlassian.com/software/jira

- Features:
  - **Task management:** to create, assign, prioritize, and track tasks
  - **Agile tools:** supports Scrum and Kanban boards, sprint planning, backlog management, and burndown charts

CI Tools - JIRA

- **Integrations:** integrates with a variety of other tools such as Confluence, Bitbucket, and various CI/CD systems
- **Customizability:** customizable to meet the needs of the team, including workflows, fields, and reports
- **Reports and Analytics:** provides detailed reports and analytics on project progress and team performance

CI Tools - JIRA

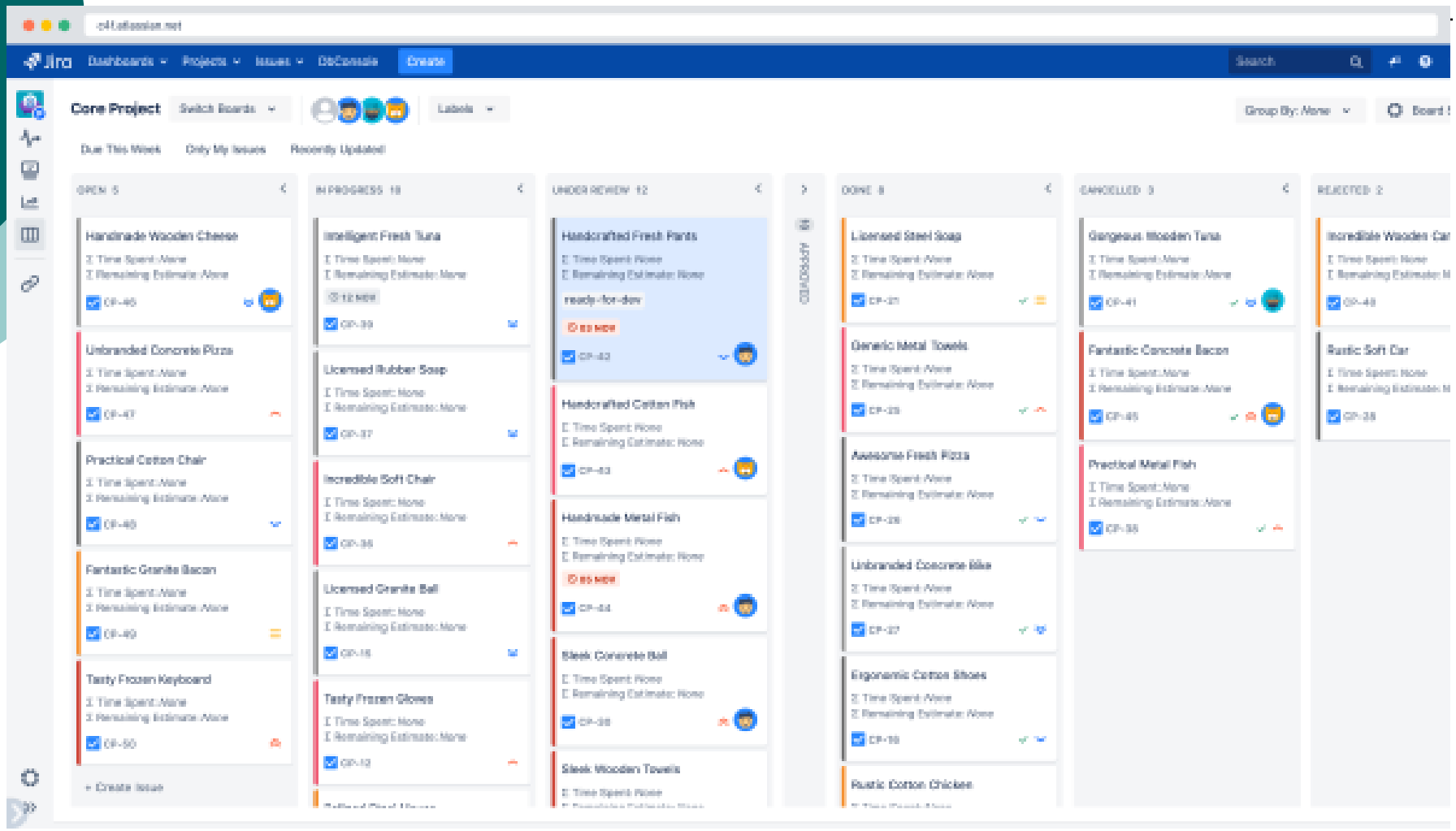


Source: https://www.atlassian.com/software/jira

CI Tools - JIRA

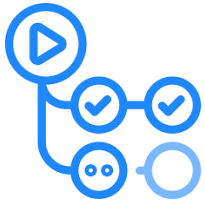
Source: https://www.atlassian.com/software/jira

CI Tools - JIRA



Forrás: https://marketplace.atlassian.com/apps/1222860/work-management-for-jira?tab=overview&hosting=datacenter

CI Tools - Github actions



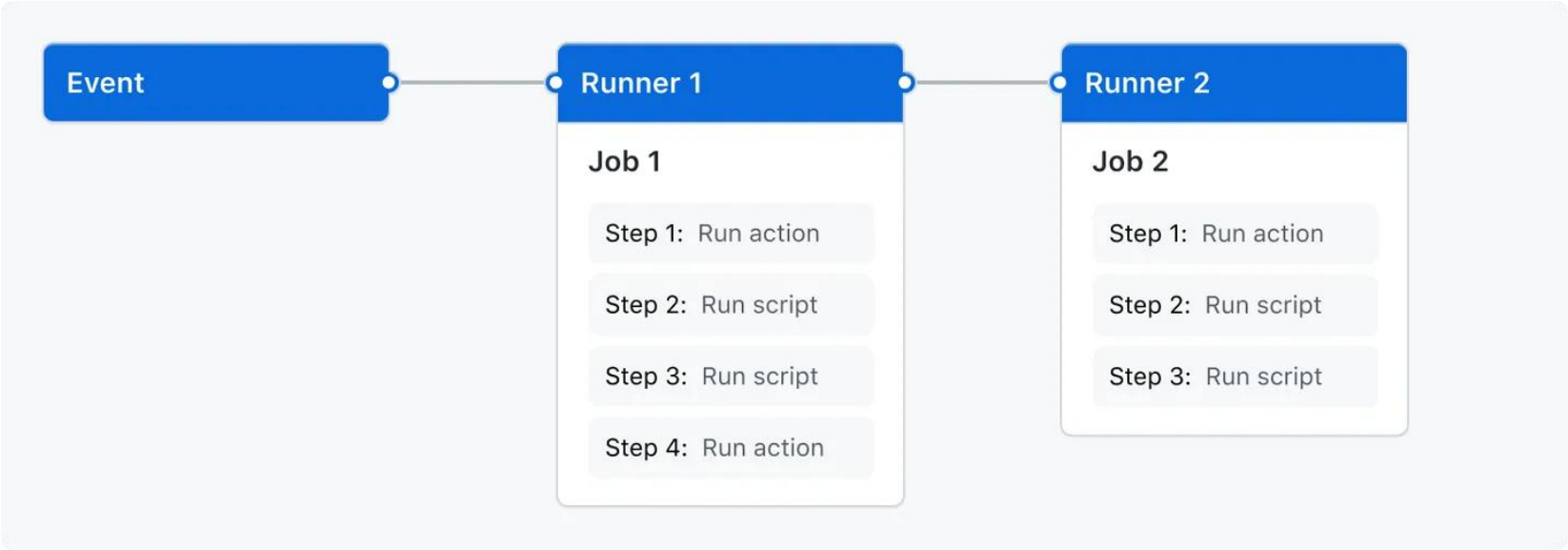
- **„Automate, customize, and execute your software development workflows right in your repository with GitHub Actions. You can discover, create, and share actions to perform any job you'd like, including CI/CD, and combine actions in a completely customized workflow”**

https://docs.github.com/en/actions

- GitHub Actions is a CI/CD platform that allowing to automate build, test, and deployment processes directly in GitHub repositories

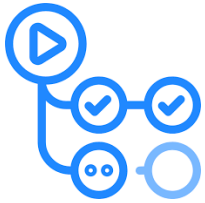
CI Tools - Github actions

- Allowing to create workflows that run automatically when certain events occur in the repository
  - e.g. when creating a pull request or pushing a commit



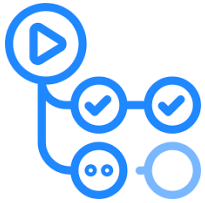
Source: https://docs.github.com/en/actions/about-github-actions/understanding-github-actions

CI Tools - Github actions



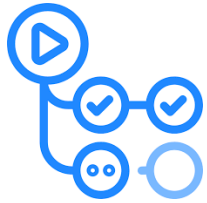
- **Features:**
  - **Automation:** allowing to automate software development processes, including build, test, and deployment
  - **CI/CD: s**upports continuous integration and continuous delivery so that code changes can be quickly and reliably delivered to the production environment
  - **Customizability:** fully customizable workflows can be created that perform different tasks, such as adding tags to new issues or installing applications after each release

CI Tools - Github actions



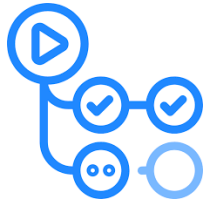
- **Features:**
  - **Integration:** integrates with many other tools and services such as Docker, AWS, Azure, etc.
  - **Ease of use:** defines workflows in YAML files that are located in the repository's .github/workflows directory
    - YAML: Yet Another Markup Language (or Ain't Markup Language), a data serialization format that can be used to create configuration files and implement data exchange
      - simple syntax, hierarchical structure, support for data types, many supported platforms/technologies

CI Tools - Github actions



- **Definitions:**
  - **Workflows:** a configurable automated process that runs one or more jobs. Workflows are defined in YAML files and run in response to various events, such as creating a pull request or pushing a commit
  - **Events:** activities in the repository that trigger the workflow to run. For example, creating a pull request or opening an issue
  - **Jobs:** a sequence of steps in a workflow that runs on the same runner. Each step is run by a shell script or an action

CI Tools - Github actions



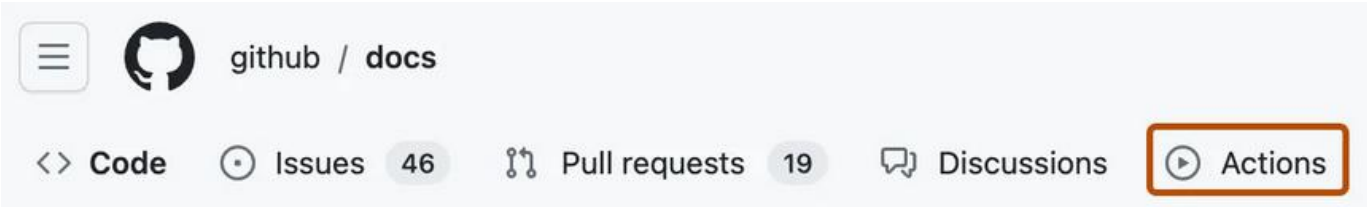
- **Example:**

```
name: CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'
    - name: Install dependencies
      run: npm install
    - name: Run tests
      run: npm test
```

- Runs automatically when a commit is pushed to the repository
  - checks the code, installs dependencies, and runs tests

CI Tools - Github actions

- **Example with Maven:**



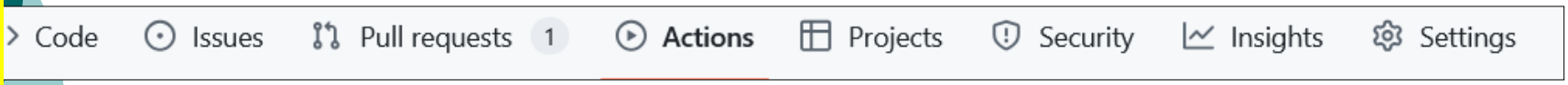

```
steps:
  - uses: actions/checkout@v4
  - uses: actions/setup-java@v4
    with:
      java-version: '17'
      distribution: 'temurin'
  - run: mvn --batch-mode --update-
snapshots verify
  - Run: MKDIR Staging & CP
Target/*.jar Staging
  - uses: actions/upload-artifact@v4
    with:
      name: Package
      path: staging
```

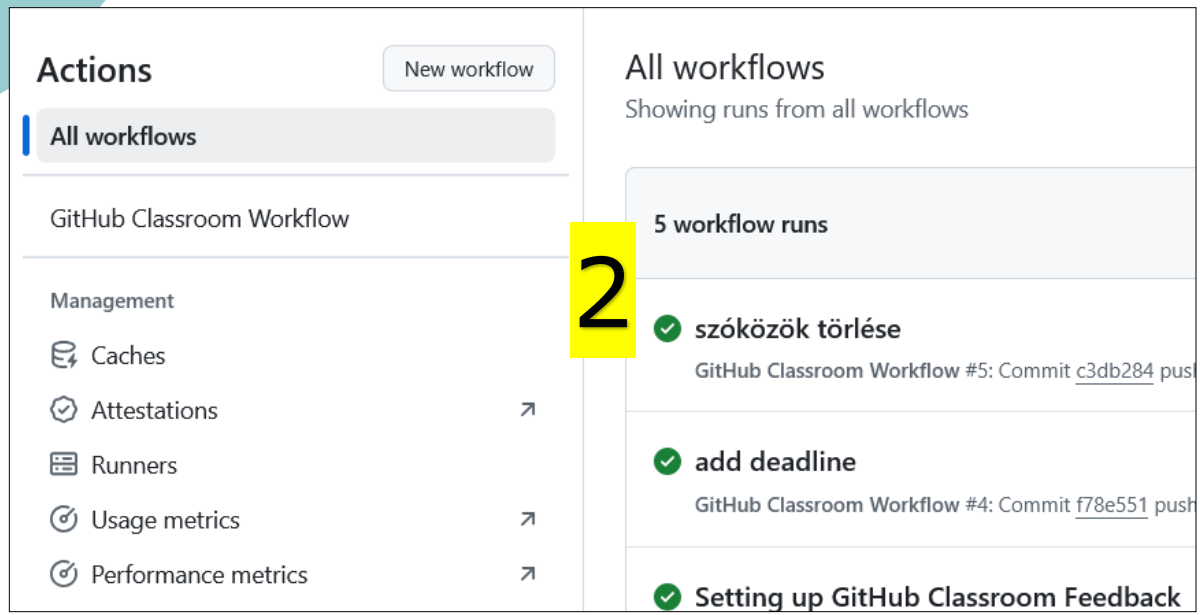
- New workflow
- "Java with Maven"
- Configure
- Commit changes
- maven.yml file in .github/workflows directory

- automatically downloads the repository
- sets up the Java environment
- run Maven tests
- wrap the build result
- then uploads it as an artifact

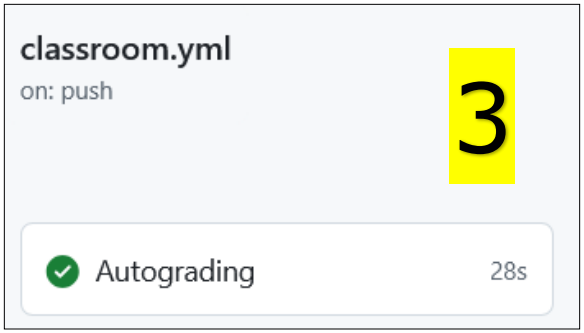
CI Tools - Github actions

1

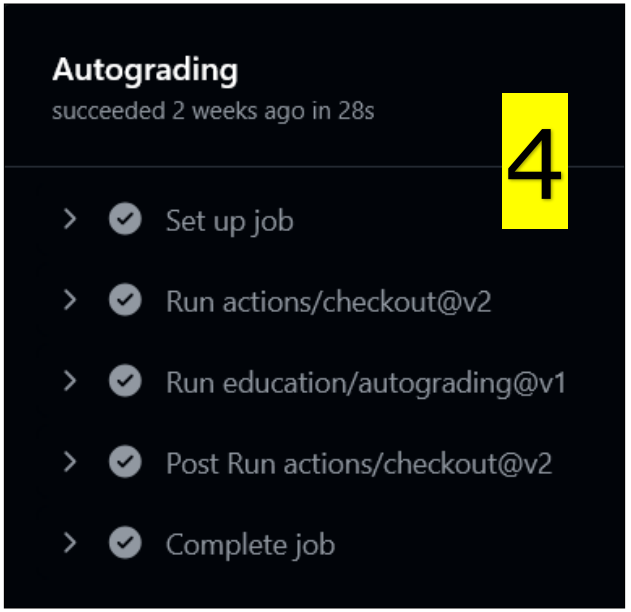




2



3



4

CI Tools – Jenkins

- Jenkins...

Thank you for your attention!