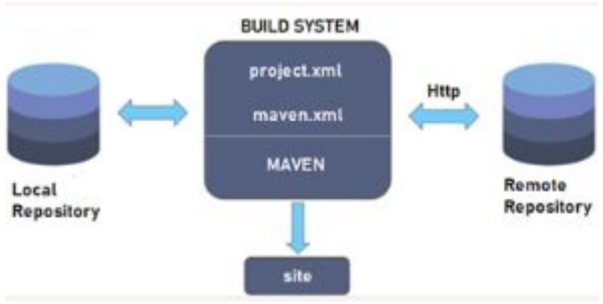
# Introduction to Maven

"Exploring Maven: Building, Managing, and Automating Java Projects"

By: Daniel Mercado Cavazos

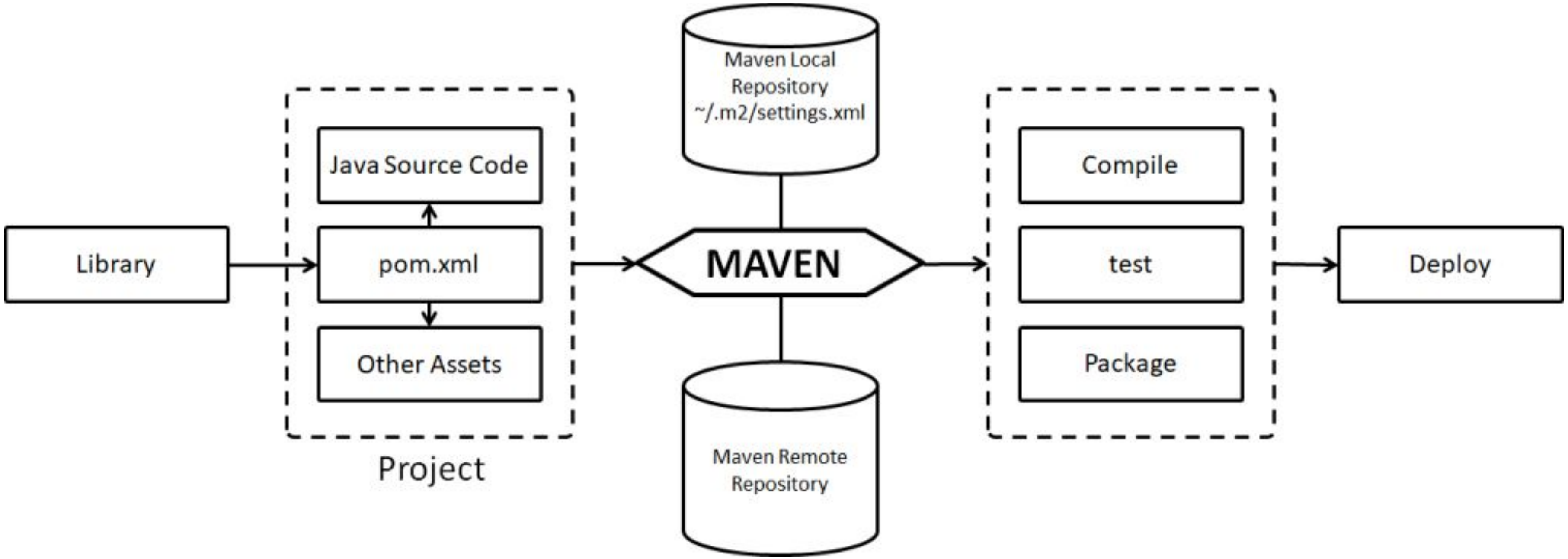
## What is Maven?

**Maven** is a build automation tool used primarily for Java projects. It simplifies the process of building, managing, and deploying software.



## What is Maven's purpose?

- **Project Management**: Handles the configuration and dependencies for your project.
- **Build Automation**: Automates repetitive tasks like compiling code, running tests, and packaging applications.


Maven Local
Repository
~/.m2/settings.xml
Java Source Code
Library
pom.xml
Other Assets
Project
MAVEN
Maven Remote
Repository
Compile
test
Package
Deploy

## Why Use Maven?

**Consistency**:

- **Standardized Project Structure**: Maven enforces a standard directory layout and naming conventions, making it easier to manage and understand projects.
- **Uniform Build Process**: Provides a consistent approach to building projects, which is especially useful in team environments.

**Dependency Management**:

- **Automatic Dependency Resolution**: Maven can automatically download and manage the libraries your project depends on from a central repository.
- **Version Control**: Easily handle different versions of dependencies and ensure compatibility.

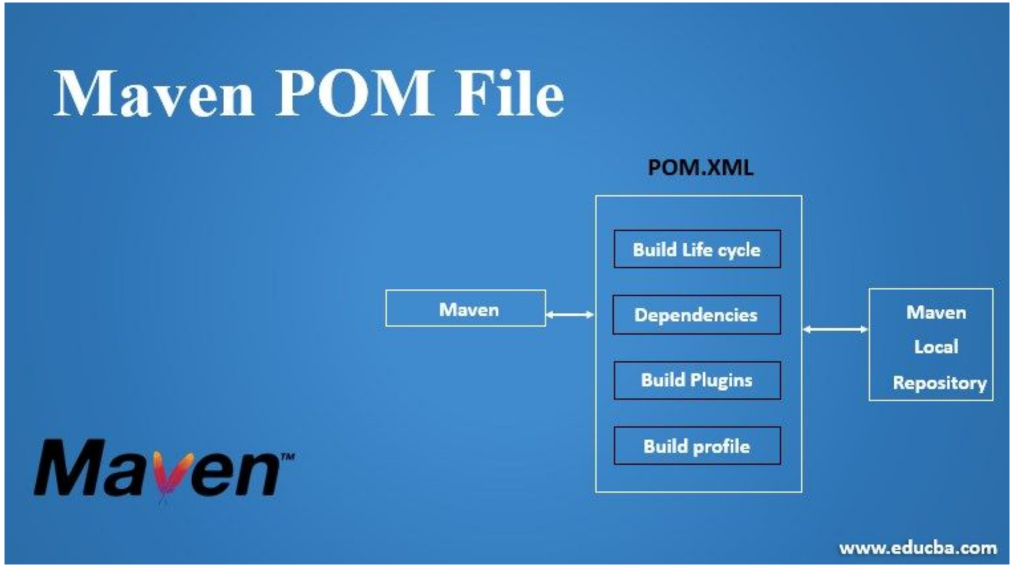
**Build Automation**:

- **Streamlined Builds**: Automates common tasks such as compiling code, running tests, and packaging applications into deployable artifacts.
- **Customizable Builds**: Allows you to define and customize build processes through plugins.

## How Maven Work?

**Project Object Model (POM)**:

- **Definition**: The `pom.xml` file is the core of a Maven project. It contains information about the project, configuration details, and dependencies.
- **Components**: Includes project metadata, dependencies, build configurations, and plugins.



**Phases**:

- **Build Lifecycle**: Maven defines a sequence of steps for building a project. Common phases include:
  - `compile`: Compiles the source code.
  - `test`: Runs tests on the compiled code.
  - `package`: Packages the compiled code into a JAR or WAR file.
  - `install`: Installs the package into the local repository.
  - `deploy`: Deploys the package to a remote repository.

**Plugins**:

- **Function**: Extend Maven's functionality by adding tasks or capabilities (e.g., compiling code, generating documentation).
- **Configuration**: Defined in the `pom.xml` file, specifying which plugins to use and how to configure them.

## Benefits of Using Maven

**Efficient Dependency Management**:

- **Automatic Handling**: Maven automatically downloads and manages project dependencies from remote repositories.
- **Version Control**: Simplifies managing different versions of libraries and ensures compatibility.

**Consistent Project Structure**:

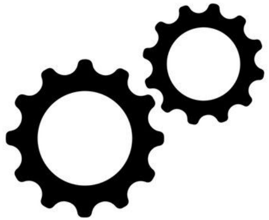
- **Standard Layout**: Enforces a standard directory layout, making it easier for teams to understand and navigate projects.
- **Predictable Builds**: Ensures that builds are consistent across different environments.

**Easy Integration with CI/CD Tools**:

- **Continuous Integration**: Integrates seamlessly with CI tools like Jenkins, Travis CI, and others to automate builds and deployments.
- **Continuous Delivery**: Facilitates automated deployment processes, improving workflow efficiency.

**Extensible and Customizable**:

- **Plugins**: Allows the use of various plugins to extend Maven’s capabilities (e.g., code analysis, documentation generation).
- **Custom Build Goals**: You can define custom build goals and processes tailored to your project’s needs.



## Conclusión

- **Maven Overview**: Recap that Maven is a powerful build automation tool for managing and building Java projects.
- **Core Features**: Reiterate Maven’s key features such as dependency management, standardized project structure, and build automation.
- **Benefits**: Highlight the benefits of using Maven, including efficiency, consistency, and ease of integration with CI/CD tools