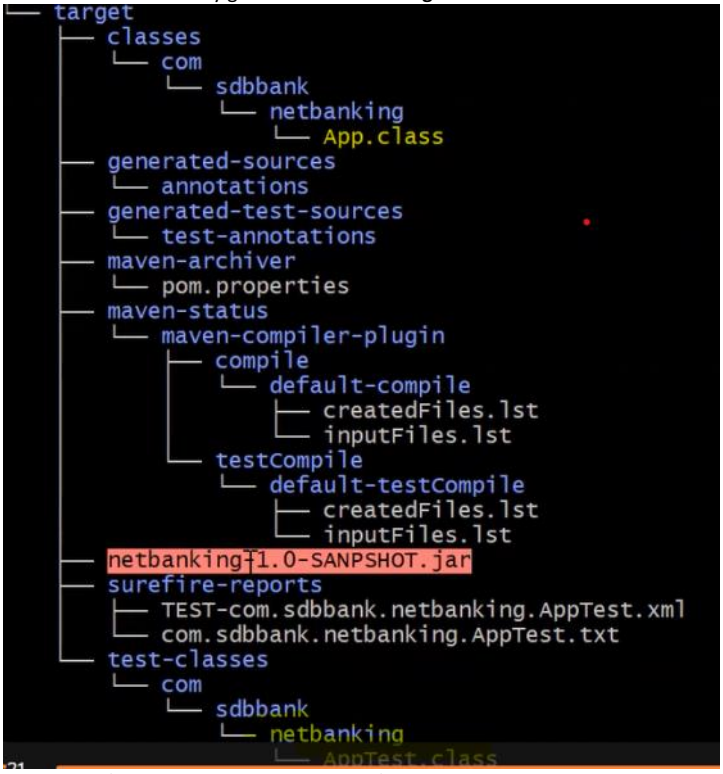
Maven-Day-2

23 November 2023 18:38

- Now the we have code ready & pushed into GitHub
  - **Will you deploy the source code that is available in GitHub as it is in DEV/QA/PROD environments?**
    No
  - We have to convert the source code into binary package. How you do it?
    Go to the repo & run command
    mvn install
  - This command will

    - **Download required dependencies from internet - Without maven it will be very difficult to download libaries**
    - **Generate binary file (.jar)**
  - Now one more directory got created called **'target'**



    - **Here functional code related .class files we can see**
    - **Also we can see unit test code related .class files**
    - **jar file created based out of .class files**
  - The generated .jar we will deploy into dev/qa/prod Tomcat/Kubernetes/Cloud foundry environments.

- Let's understand **MAVEN Build Lifecycle**
  - In Maven, the build process organized in series of well-defined phases & the sequence of these phases referred as Build Lifecycle.

- 

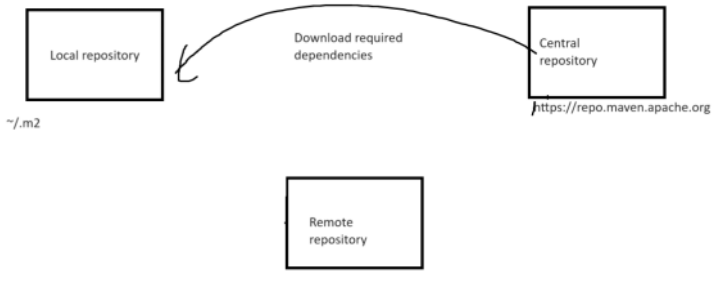
8 Phases of the Default Maven Lifecycle

  - If you see above diagram there are multiple phases in build process in sequence to generate & deploy the binaries.
  - Let's discuss how each phase will help
    - Validate - Validate the directory structure of the project.
      mvn validate
      Check the directory structure created correctly or not.
    - Compile - Convert the .java files into .class files & we can see these files in target folder
      mvn compile
      Here in target folder we can .java files are converted into .class files
    - Package - Will generate the binary file .jar based out of .class files
      mvn package
      Here in target folder we can see .jar file
    - Verify - Verify that the project is valid and meets the quality standards, Code coverage should be > 80%
      mvn verify
      - Suppose if you are a developer & wrote 100 java files for functionality but written only 2 java files for unit test.
      - Which means only 2% of code is tested at developer side.
      - We can restrict the developers by making build failed until they wrote test code more than 80% by using **Jacoco** plugin in pom.xml
    - Install - Copy the generated .jar file from target folder to local repository(~/.m2).
    - Deploy - It copies the packaged .jar file to the remote repository for sharing it with other developers
  - Along with these goals we can use clean command like
    mvn clean install

    Clean - Will delete the target folder before fresh build process started.

- Next we will understand types of repositories in maven, so we will come to know install & deploy commands clearly.

- In maven we mainly 3 - Kind of repositories,
  - **Local repository**
  - **Central repository**
  - **Remote repository**

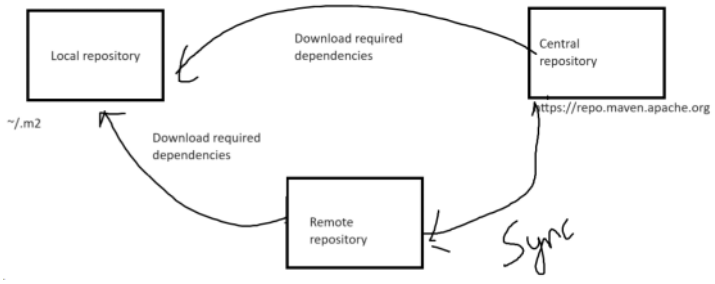


- Whenever developers run any maven command like(mvn install/deploy) maven connects to maven central repository which is present in Internet & downloads the required binaries into ~/.m2 folder on the machine to complete build process.

- The binaries are copied remote repository to local repository only at first time & next time onwards maven will consumes the dependencies from local repositories from local repository.

- Let's remove the files are downloaded in local maven repo & run **mvn install** command
  Here we can see binaries are downloaded from maven central repo to the local repository.

- Now again run **mvn install** command, this time see binaries are downloading & it's getting consuming from the local repository.

- 

- Some organizations will not allow to connect the maven central repository & download binaries.
  In these situations maven will connect to remote repositories that are create in the organization level. Maven remote repository is in sync with the central repository.

- Normally we tell maven where to download binaries either from central/remote repo based on configurations in settings.xml(%MAVEN_HOME% /conf/)

```
<mirrors>
  <!-- mirror
   | Specifies a repository mirror site to use instead of a given repository. The repository that this mirror serves.
   -->
  <mirror>
   <id>central</id>
   <url>https://repo.maven.apache.org/maven2</url>
   <mirrorOf>central</mirrorOf>
  </mirror>
  <!-- ... other mirrors ... -->
</mirrors>
```

  If we don;'t have above configufration in settings.xml by default we are using central repo for downloading binaries.

- If we want to download binaries from custom-central repo url

```
<mirror>
 <id>central-custom</id>
 <url>custom-https-url</url>
 <mirrorOf>central</mirrorOf>
</mirror>
```

  <mirrorOf>: Specifies which repository or group of repositories this mirror should replace. In this case, it's set to "central," indicating that it's a mirror for the Maven Central Repository.

► **mvn deploy explain practically during Jfrog setup. - Try this during Jfrog session**

1. Setup maven repository in Jfrog

2. Configure Deployment in pom.xml: In your project's pom.xml file, you need to configure the distribution management section to specify the URL and authentication details for the remote repository. Here's an example:

```
<distributionManagement>
  <repository>
    <id>your-repo-id</id>
    <url>https://your.repository.url/releases</url>
  </repository>
  <snapshotRepository>
    <id>your-snapshot-repo-id</id>
    <url>https://your.repository.url/snapshots</url>
  </snapshotRepository>
</distributionManagement>
```

  Replace your-repo-id, https://your.repository.url/releases, your-snapshot-repo-id, and https://your.repository.url/snapshots with your repository details. Make sure to use the correct URL for release and snapshot repositories.

3. Configure Authentication: Maven needs authentication details to deploy artifacts. You can configure these details in the settings.xml file, which is typically located in the <Maven_Home>/conf directory. Alternatively, you can configure authentication directly in the pom.xml file (although this is less secure):

```
<servers>
  <server>
    <id>your-repo-id</id>
    <username>your-username</username>
    <password>your-password</password>
  </server>
  <server>
    <id>your-snapshot-repo-id</id>
    <username>your-username</username>
    <password>your-password</password>
  </server>
</servers>
```

Replace your-repo-id, your-username, and your-password with your repository and authentication details.

4. Run mvn deploy: Now, you can run the following command in your project's root directory to deploy the artifacts:

```
mvn deploy
```

Maven will compile your project, run tests, and if successful, deploy the artifacts to the specified remote repository.

Home-work

1. Download the binaries from custom central maven repository
2. Build any java application & generate binaries & automatically copy it to local repository on fresh ec2 machine
3. Explain the Maven Build Lifecycle.
4. Explain the difference between a Maven Snapshot and a Release.
    - A Maven Snapshot is a version of a project under development, typically not intended for release. A Release is a stable version of a project that is considered ready for production use.
5. Deploy artifacts to remote repositories

1. Explain Maven life-cycle
2. Explain with scenario how plugins downloaded from the Jfrog remote repositories
3. Install Jenkins
4. Install Git & Maven on server
5. Integrate Git with Jenkinsfile
6. Integrate Maven with jenkins
7. Create maven project
8. Create pipeline project
9. Execute the pipeline