Maven

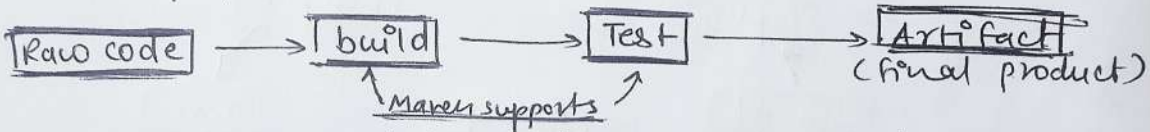
(home path - .M2 folder)

- developed by apache software foundation in 2004 written on Java programming language. (supports - Java 1.8.0)
- It is a popular build automation tool for Java based projects that helps, manage and organise the software development process
- It provides a standard way to define project dependencies build processes and project structure. also called as project management tool.
- Maven use declarative approach to configuration and relies on POM.xml file to describe the project's structure, dependencies and build process.

POM : project object model

XML : Extensible markup language.

- It is opensource, free and offers a vast repository of libraries and plugins that can be downloaded and integrated in the project.
- Developer will give us the Raw code we can't use or deploy that Raw code directly first we have to build (compile) and test the Raw code then it will convert into artifact (final product) which can be used to deploy.



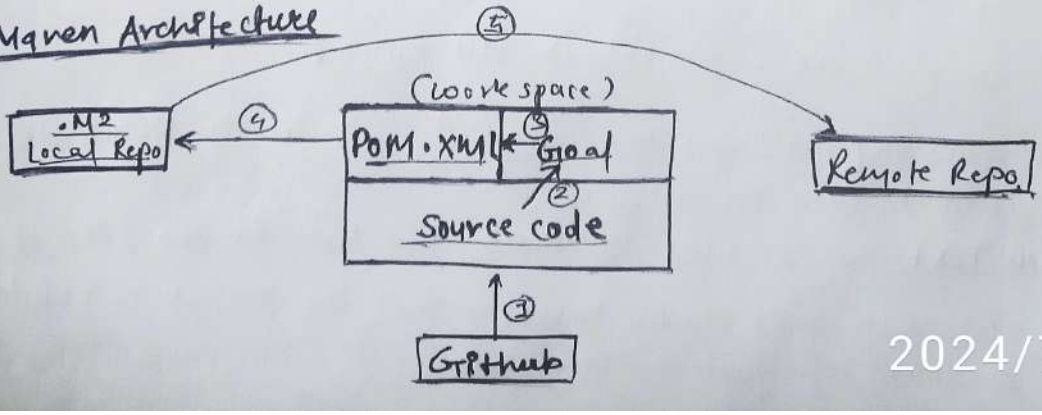
#Artifact

→ It contains both compiled code and resource used to compiled them.
→ final product of the code which can be deployed.

- Jar file (Java Archive) — Contains Backend code. (Java class and associate metadata)
- war file (web Archive) — Contains frontend + Backend code. (Jar files, Jare servlets, xml files.)
- Ear file (Enterprize Archive) — Contain Jar + war

- If we want to change backend code we deploy Jar file and if we want to change frontend and backend code we deploy war file
- In Real time we mainly deploy war (web archive) file.

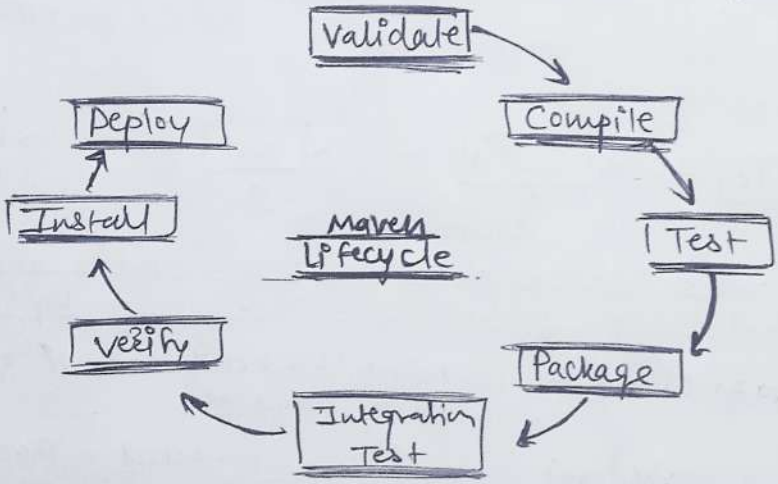
• Maven Architecture



2024/7/12 04:02

1. — Downloading source code from github.
2. — using goals commands (Compile, test, install) we work on src code
3. — goal will refer the pom.xml file because it has tool's information
4. — Pom.xml will ask local Repo for the plugins
5. — now Local Repo will download the plugins from Remote Repo and integrate it in projects.

- **Plugins** — It is small s/w which make our work automate.
  - Instead of downloading tools we can download plugins as it work same as tools and are light weighted.
  - this plugins will download automatically when we run the goal command
- **Maven Lifecycle** — The default maven lifecycle consist of 8 major steps.
  - **Goals** — a commands used to run the task



- **validate** — This step validates if the structure is correct. - It check if all dependencies are downloaded and available in local Repo.
- **Compile** — It compiles the src code into executable code ie .java file to .class file and store class file in target folder
- **Test** — It runs the unit test for the project
- **Package** — This step packages the complied code in a distributed format like Jar or war.
- **Integration Test** — It runs the integration test for the project.
- **verify** — This step runs checks to verify that the project is valid and meet the quality standards.

- **Install** - This step installs the packaged code to the local maven Repository (.m2)
- **Deploy** - It copies the packaged code to remote repository for sharing it with other developers.

Commands

[MVN compiles] - compile code from .java to .class file [src fold] → [target fold]

[MVN Test] - Test the code test.java to test.class [test fold] → [target fold]

[mvn test-compiles] - compile + test

[mvn package] - use to create the artifact [target fold]

[mvn install] - copy artifact [target fold] to [.m2 folder]

[mvn clean] - delete the target folder

[mvn clean package] - delete old version and do steps from compile → install

[mvn site] - generate the site/project documentation

[mvn validate] - validate the project's pom and configuration.

[mvn deploy] - copies the jar/war file to remote Repo

Initial command

1. Create EC2 Instance and connect.
2. yum install git java-1.8.0-openjdk maven tree -y (shows project structure)
3. yum pull.
4. git clone http://github.com/devopsbyrahaul/Jenkins-java-project (Project URL / GitHub link)
5. cd jenkin-java-projects.

[tree] - shows the project structure.

[ll /target] - Contain .jar/.war file.

2024/7/12 04:03

- **Maven vs Ant**

1. Maven is build and project management tool, Ant is only build tool
2. Maven has Pom.xml, Ant has build.xml
3. Maven has Lifecycle, Ant haven't.
4. Maven Plugins are reusable, Ant scripts are not reusable
5. Maven is declarative, Ant is procedural

- **Programming** → **Build**
  - Java → Maven
  - Python → Gradle
  - .NET → VS code
  - C, C# → Make File
  - node.js → npm (node package manager)

Jenkins

- developed by sun micro system employe Kohsuke Kawaguchi
- Initially named as Hudson (Paid)
- later sun micro system was bought by the orcale and renamed Hudson to Jenkins (free).
- It is written on java and supports java 11/17/18
  - use to schedule jobs and integrating tools in automation.
- free / open source / platform independent / community support
- [- default path /var/lib/jenkins]
- Port-No. 8080
- **Alternatives** - bamboo, Go CI, circle CI, GitLab, Travis, Harness.

- Jenkis is CI/CD tool (Reality - Jenkin can only do CI)

Continuous Integration (CI) - continuous build and continuous test (old code with new code)

- It is a practice where developers regularly add/merge new code into old code present in central Remote Repository
- And devops Engineer continuously/Regularly build and test it using continuous integration tool called Jenkins.
- It automate the CI/CD process and saves time.

Continuous delivery / deployment

- a process that automate the s/w release process, including building testing, configuring and deploying code to production enviornment.
- The goal of CD is to make it easy as possible to deploy new code and to ensure that there's minimal efforts involved.
- Both continuous delivery and deployment are same except the final step of deployment. where
  - in continuous delivery, a manual decision is required to deploy application to production.
  - in continuous deployment, the decision/process is completely automate to deploy the application to production.

• Enviornment (isolated and controlled space in s/w development).

① Pre production Enviornment / non Prod Env.

- Dev : developer
- QA : tester
- UAT : client

- **Pre Prod Env** is a controlled and isolated space in s/w development where developers and quality assurance teams can test and validate s/w before it goes live.
- First developer will write the code and will test it in local server later passes it to QA team to testing and validating the appl^n then QA team will pass it to client satisfaction then to the live prod Enviornment here user is going to use the appl^n.

② Live Enviornment

Prod : users

- **Live prod Env** is space where s/w goes live to user's use. Where Appln is use by people in real time.

NOTE : To do continuous Integration, Delivery and deployment we need to have pipeline and to create pipeline we need jenkins

Pipeline - series of processes which are linked with each other and are executed step by step.

[Plan → code → build → test → deploy → operate → monitor]

[code → compile → test → artifact → deploy]

Practical (setting up jenkins on server)

1. create EC2 instance which includes all traffic in sg and connect.
2. install git, java and maven
   + [yum install git java-1.8.0-openjdk maven tree -y].
3. Getting Repo (browse jenkins.io → go to download → in stable release → go to redhat)
   - need to download this repo using below commands because this Repo will consist of some important plugins and required keys for the jenkins server.

4. Download java and jenkins
   + [amazon-linux-extras install java-openjdk11 -y]
     (instead of yum we will use this to download java)
   + [yum install jenkins -y]
   + [update-alternatives --config java]
     - as we download java 1.8.0 first on server by default this version will on server but for jenkins we need to run java 11 we installed.
     - This command is used to change the java version from 1.8.0 to 11/17.

- when we download any service by default it will be on stopped state we need to start it using some commands
- as jenkin is also the services which need to be start

⑤ Restarting jenkins

+ [systemctl start jenkins.service]
+ [systemctl status jenkins.services]

Connect

① public ip address : 8080 (on browser)

② It will open the jenkins authentication page, copy the path given on page and use it in server terminal (cat /var/lib/jenkins/secrets/initialAdminPassword) you will get one key use it for login purpose

③ new page will open for installing plugins click on "Install suggested plugins" button it will download plugins like Ant, Gradle, Git, Github branch source, pipeline, etc. - - -.

④ now create first admin user will open give the credentials like username, email and passkey

⑤ jenkins home / dashboard will open.

JOB - It is used to perform task, to do any work / task in jenkin we need create job.
- If we want to do continuous integration, delivery, deployment we need to create the job for it
- to run command we need to select execute shell on build steps.

• build now :- to run job
• workspace :- place where our job output will come ie jar file in target folder.

default workspace path /var/lib/jenkins/workspace.

• configuration -

- creating a Job

New Item → name PT → select freestyle → select ok → Go to Source code management (scm) → select GIPt → Past Remote Repo url of project → go to build steps → select Execute shell → enter mvn clean package command → select save → click on build now.

- when developer add the new code to github Repo you just click on build now to CI entire new and old code.

custom workspace

- In Real time we don't save our output on default workspace.

[cd] - first come to home directory.

[mkdir Sukant] → custom workspace

[cd /sukant/]

go to configuration (jenkin home page) → go to advance → On use custom workspace → paste the customeworkspace → workspace path → save

[/root/sukant].

- now we have to give permission to customeworkspace [/root/sukant]

- changing ownship (chown)

[chown jenkins : jenkins /root]

[chown jenkins : jenkins /root/sukant]

(jenkin user) (jenkins group)

Variable - It is used to stores values that are going to change frequently.
Ex - date, season.

- **Type of variables in jenkins**

① **User defined variables** (defined by the user)
- **Local variables** (defined within job)
- **Global variables** (defined outside job)

② **Jenkins Env variables** (predefined by Jenkins)

① **User defined variables** - the variables which are declared and defined by the users in jenkins Environment.

- **Types of user defined variable**

(a) **Local Variables** - Variables which are declared and defined within the jobs are termed as local variables.
- The scope of this variables are within the job only i.e can be used in the job in which it is declared.

New Item → Name → freestyle → ok → build steps → Shell Execute

variable
name = "ranam" ← value
echo " my name is $name "
echo " $name likes in hyd "

- can't give space b/w variable, assigning operator or value or else where it will give syntax error.

(b) **Global variables** - variables which are declared and defined outside the jobs termed as Global variables.
- The scope of this variable is upto all the jobs created in jenkins i.e can be used in all jobs as it is declared outside jobs.

dashboard → manage jenkins → system → Global properties
→ Environment variables → add → name : value

- **Shadowing** - Local variable shadowed the scope of Global variable.

- some values can't be vary build by build like build no., time, etc so, such values are predefined in jenkins already.

② **Jenkins Env variables** - This variables are predefined by the jenkins.
- these variables can change/vary build to build.
- these variables will be always in upper case by default.
- these variables can be defined only once.

Ex $BUILD_NUMBER , $BUILD_ID - - -

- [printenv] - print all the jenkins enviornment variables.

• Admin task

ⓐ changing port No.

- [find / -name "jenkins.service"]
- [vi /usr/lib/systemd/system/jenkins.service]
- [line :7o change 8080 → other port no -]
- [save + Exit]
- [systemctl daemon-reload]
- [system ctl restart jenkins.service]

ⓑ Passwordless login in jenkins

- [find /-name "config.xml"]
- [vi /var/lib/jenkins/config.xml] false
- [line → 4 change → <use security> true </use security>]
- [systemctl restart jenkins.service]

ⓒ Jenkins server crashed

- Stop the main jenkins server EC2 instance then increate the instance type and restart the server and services.

Build Executors

- An Executor is a slot that allow a job to run **on** agent.
- No. of executors determines the no. of concurrent build execution.
- By default jenkins have two build executors ie we can run two build parallely.

Job → configuration → general → Execute concurrent build

- Initially jenkins will execute all build sequentially only after enabling this option jenkins can build two build concurrently.
- now to increate the no. of build executor we can flow following step but it is not recommended because it can increase the load which can result into crashing the server.

2024/7/12 04:07

NOTE - Everytime we stop the server the services on that server will also get stopped as jenkins is also the service so whenever the server (EC2) get stopped jenkins get stopped too. so we have to always start it after the start the server.

- [chkconfig jenkins on/off] ← It will enable the setting that make jenkins automatically start when the server get start.

It is internally directed to this command

- [systemctl enable/disable jenkins.service]

 # Pipeline - It is series of processes which are internally linked with each other and are executed step by step.

Raw code → build → Test → Artifact → Deploy

- **Why to use** - (1) to automate the task
  (2) to have clarity of the stage (compile, test, package, deploy)

- **Type of pipeline** - (1) **Declarative**
  - use a domain-specific language to define stages, steps and directives
  - more human friendly and easier to understand
  - allow you to skip stages using the when block.

(2) **scripted**
  - use the groovy scripting language to provide more flexibility and customize
  - They allow more error handling and recovery mechanism
  - you cannot skip entire stage but can skip a login within a stage using if statement.

- **pipeline syntax** — indicate that it is pipeline script.

```
pipeline {
    agent any
    stages {
        stage ('stagename') {
            steps {
                I
            }
        }
    }
}
```

agent any ← indicate can run on any agent (node)

a machine/container that connects to master server and execute tasks as directed by master server.

stages { ← here is where all the work happened. stages can contain n no. of stage within it.

stage ('stagename') { ← here we write all over stages work and one stage can have only one steps in it.

steps { I } ← here we write our various commands ('git' / (or) sh ' / (or) echo " ")

- **short for pipeline syntax** (PASSS)

P	Pipeline
A	agent
S	stages
S	stage
S	steps

```
Pipeline {
    agent any
    environment {
        VERSION = '1.2.0'
    }
    parameters {
        string(name: 'ver', defaultValue: '1', description: 'lab')
        choice(name: 'ver', choices: ['1', '2'], description: 'bc')
        booleanParam(name: 'executecomp', defaultValue = true, description: '')
    }
    stages {
        stage('checkout') {
            steps {
                git 'URL of project' && echo "current version ${VERSION}"
            }
        }
        stage('compile') {
            when {
                expression {
                    Param.executecomp == true
                }
            }
            steps {
                sh 'mvn compile'
            }
        }
        stage('test & package') {
            steps {
                sh '''
                mvn test
                mvn package
                '''
            }
        }
        stage('deploy') {
            input {
                message "are the parameter correct?"
                ok "yes"
            }
            steps {
                sh 'mvn deploy'
            }
        }
    }
}
```

Whatever variable we declared here will have scope in all the stages.

used to pass information for job.

pass single input at time

pass multiple inputs at time

pass input either yes or no.

Single stage pipeline : will have only one stage.

Multi stage pipeline : will have multiple stages in it.

this will only execute when the expression block have true parameter else where this stage will be skipped entirely

- like this we can run more linux command in single sh

Input parameters : Based on user Input the pipeline is going to execute.

It will give two options of "yes" or abort the stage or pipeline

In realtime give manual approval is a good practice.

NOTE:

- To search anything on console output (ctrl + K) then type word to search)

- (**ec2-metadata --all** → show all info about EC2 instance)

2024/12/04 04:07

Master and slave

- It is used to distributes the master's builds within the slave servers.
- To reduce the load on jenkins server/master server.
- communication between slave and master happens through ssh. (Secure Socket Host)
- We need to install agent on slave server which is java 11. or the slave will not work at all.
- slave can use any platform but we can go with linux.
- Label ← It's a name given to the slaves for uniquely identifying them so that master can assign them build/task properly.
  - If we don't give the label all slave will on confused state, that who will take the build to execute.

• Setting up slave

1. Create EC2 instance and connect.
2. [amazon-linux-extras install java-openjdk11 -y]
3. [yum install git java-1.8.0-openjdk maven tree -y]

• Creating slave node

Dashboard → Manage jenkins → Node → newNode → name the node → parment agent → create

• Configuring slave node

- **Label** → way of giving the unique way to node so masters node can assign them build properly without getting any confusion.
- **Remote root directory** → /tmp : the place where slave server will store the output (.jar or .war).
- **Number of executers** → can set the build executers to set the slave's concurrent build capacity.
- **Lauch Method** → here we will select the ssh (Secure Socket Host)
- **host** → give slave server's Ip address (public can also be given).
- **credentials** → add → jenkins credential provider → make kind to ssh username with private key → username give ec2-user → in private key paste the pem key code you generate while creating slave EC2. → host verifying strategy select → non verifying strategy.
- **usage** : select "only build jobs with label expression matching this node"

Go to build → configuration → click on Restrict where this project can be run → give slave label name → save (on freestyle)

- after clicking on build now button the build will happen on slave node and in console output it will show on slave it is build and where the output is stored.

In pipeline to set slave we can do.

In pipeline we defined the slave name in pipeline syntax.

```
pipeline {
    agent {
        label 'slave label name'
    }
    stages {
        stage('linux') {
            steps {
                sh 'touch file1'
            }
        }
    }
}
```

Post Build actions - actions that are performed after the build is done.

1. always : executes always
2. success : executes when build is successfull.
3. failure : execute when the build is failed.

Syntax

```
pipeline {
    agent any
    stages {
    }
    post {
        always {
            echo " build is done"
        }
    }
}
```

will always execute dispite of the build happen or not