7/2/23:-

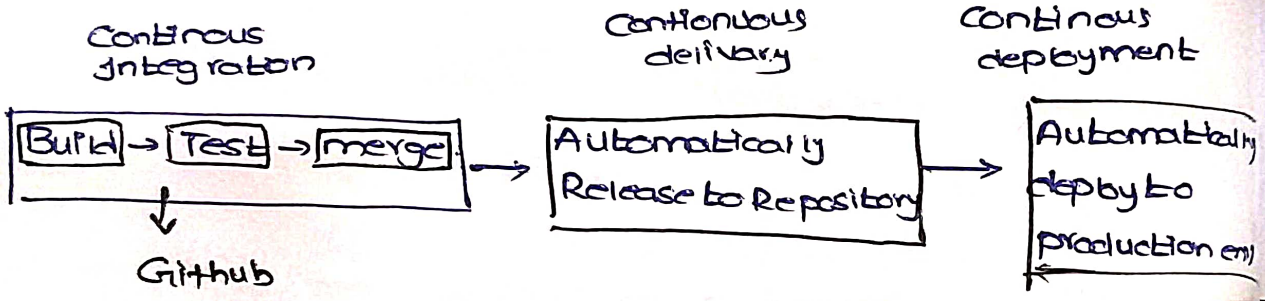
CICD:-

What is CI/CD:- CICD is a method to frequently delievery apps to customers by automation into the stages of app development.

→ main concept of CI/CD are continous Integration, continous delivary and continous deployment.

→ CI/CD is a solution to the problem integrating new code, can cause for development and operation teams

Difference b/w CI AND CD (Another and other CD)

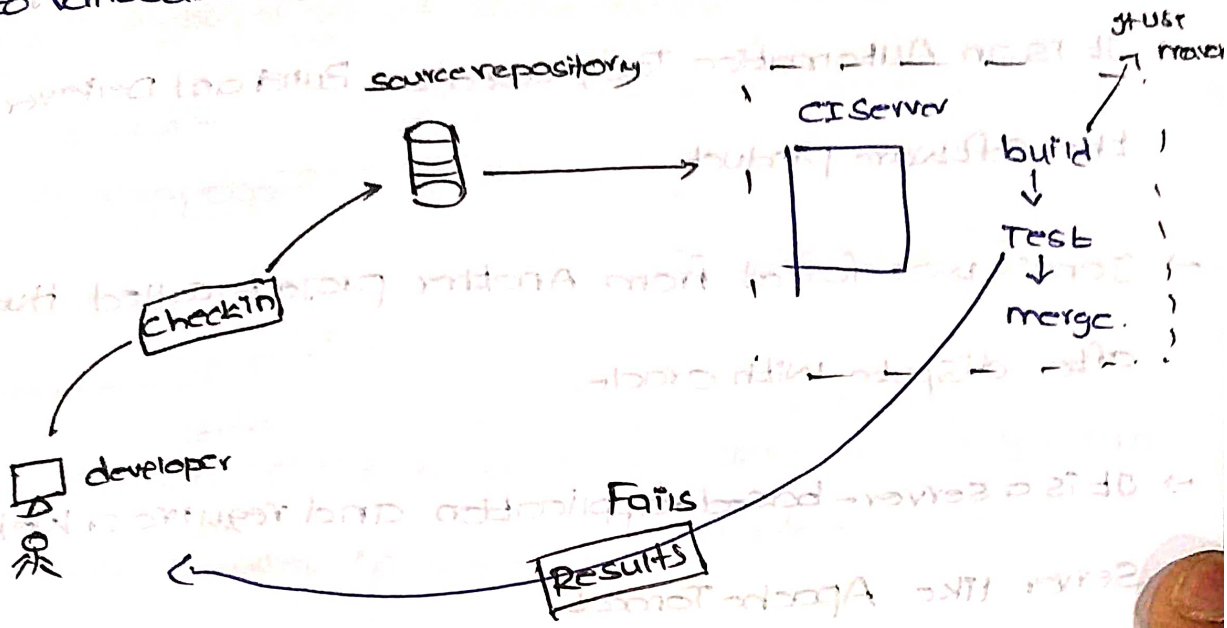


Continous Integration: [CI]

→ Developers regularly merge their code changes into a central repository, after which automated builds & tests are run.

→ CI refers to the build or integration stage of the software release process and entails both an automated component and a cultural component.

→ The key goals of CI are to find and address bugs quicker improve software quality, and reduce the time it takes to validate and release new software update.



Continous Delivary [CD]:

→ CD is the automated delivary of completed code to environments like testing and development.

→ CD provides an automated and consistent way for code to be delivered to these environment.

Continous Deployment [CD]

→ CD is the next step of continous delivary. Every change that passes the automated tests is automatically placed in production, resulting in many production deployments.

CI/CD Tools:

Jenkis

→ Jenkis is an open-source CI/CD tool written in Java.

→ It is an Automation Tool, used to Build and Deliever the software product.

→ Jenkis was forked from Another project called Hudson after dispute with oracle.

→ It is a server-based application and require a Web Server like Apache Tomcat.

→ Jenkis monitoring of repeated task which arise during the development of a project.

Eg: your team is developing a project, Jenkis will Continously test your project builds and show you errors in early stage of development.

Why use Continous Integration With Jenkins

→ Code is built and test as soon as developer commits code.

→ Jenkis will built and test code manytimes duringday

→ On Successful build, Jenkis will deploy the source into test server and notifies the deployment team

→ on Build failures, Jenkis will notify error to the development team.

→ Code is build immediately after any of the developer commits
→ Automated build and test process saving timing and reducing defects

Advantage of using Jenkis.

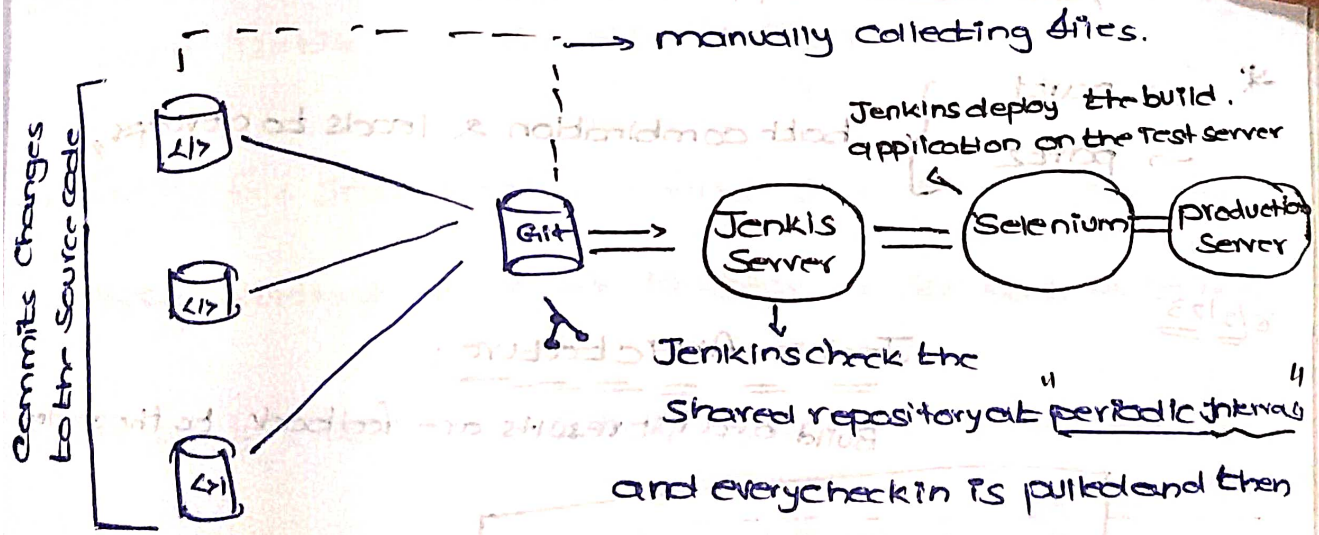
→ Jenkins is being managed by the community which is very open.

→ Jenkins has around 320 plugins published in its plugin database.

→ It support cloud-based architecture so that you can deploy Jenkins in cloud-base platform.

→ It support Docker containers, you can containerize Jenkis service.

Jenkis Architecture.



→ Automation
→ manual

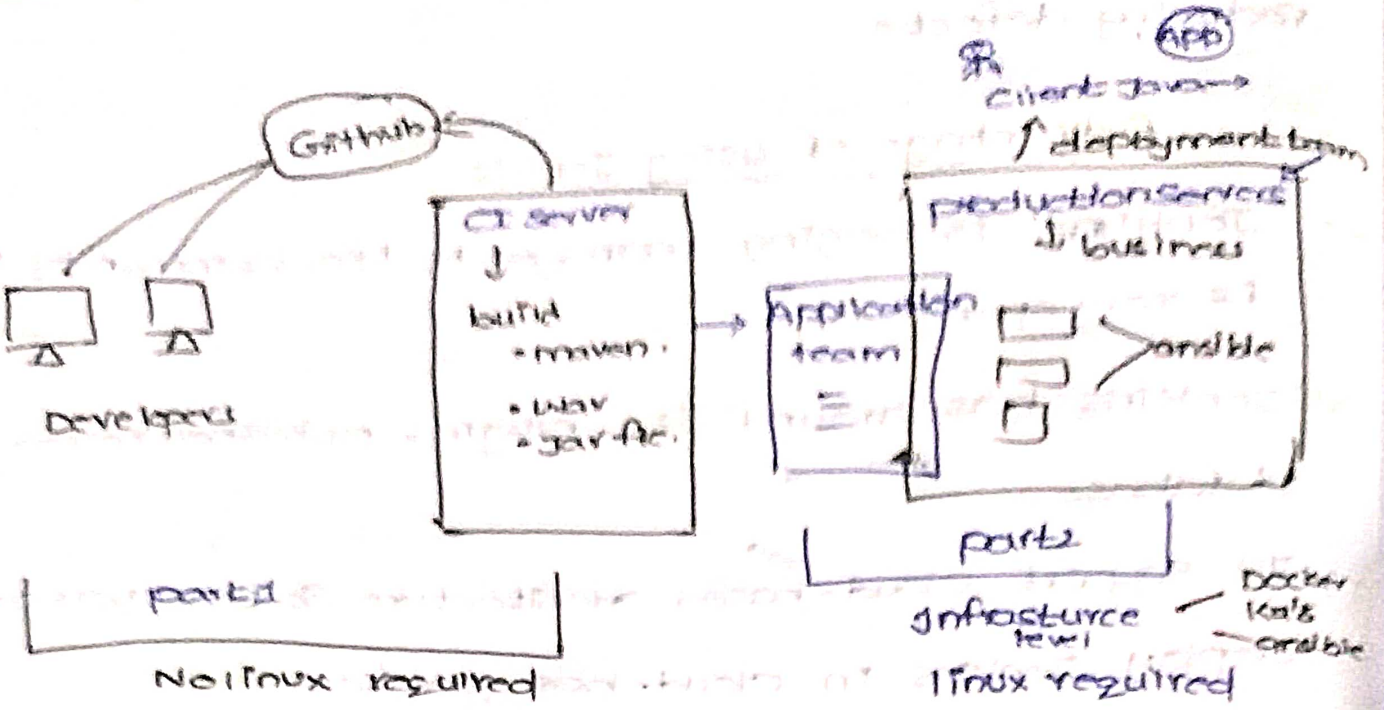
maven → build tool
Ant.
Gradle,

~~AWS~~

Workation

Real world

[deploying = installing application.



deploying → application on multiple webserver,,

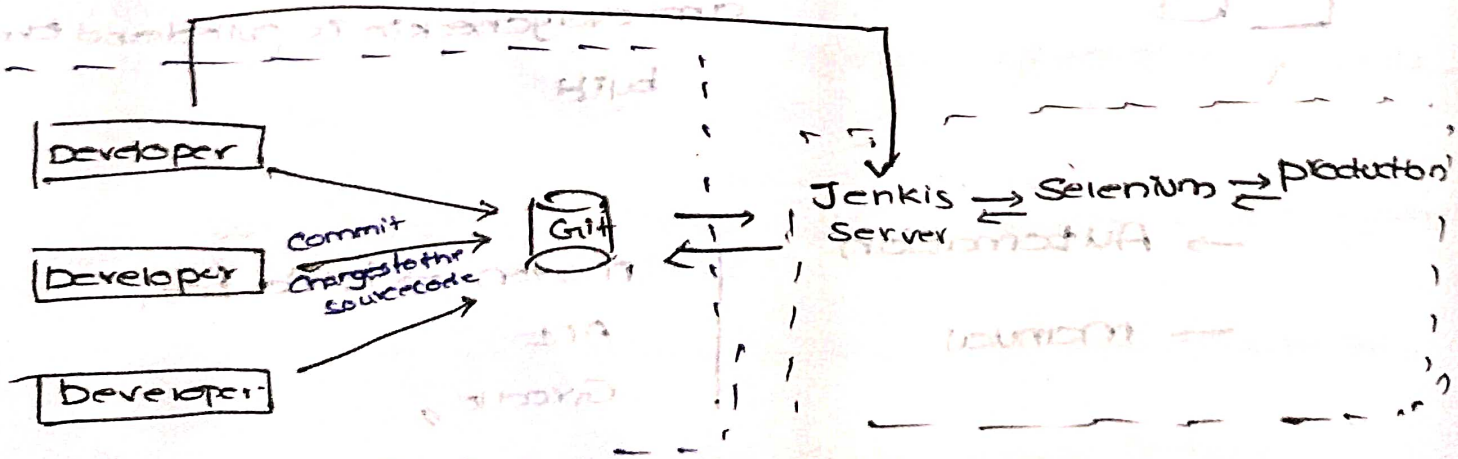
→ yum install • App

* → part1
→ part2
} both combination & leads to devops,,

8/2/23

Jenkis Architecture:-

Build and test results are feedback to the developer.



April-28 → Srikanth birthday

Jenkis Installation on Redhat server:

→ Jenkis installer are available for several linux distributions
→ debian/ubuntu, Fedora, Red Hat/centos.

Minimum H/W Requ:

256MB of RAM
1GB of drive space (10GB is a recommended minimum if running jenkins as a Docker container)

Recommended Hardware configuration for a small team:

4GB+ of RAM.
50GB+ of drive space.

Software requirements:

Java 8/11 runtime environment are supported in both 32-bit and 64-bit version.

→ Goto Ec2 Instance → Launch Instances → Jenkins
Select Redhat linux → Keypair → SSH port → Launch.

→ Add port : 8080 → Security group → Inbound →
Custom Tcp → 8080 (Tomcat) → Anywhere.
Save rules. (Jenkis required Tomcat)

→ Connect Instance → ec2-user

→ Sudo -i

Jenkis Installation on Redhat Server.

→ Jenkis installer are available for several linux distributions
→ debian/ubuntu, Feebra, RedHat/centos.

minimum H/W Requ:

256MB of RAM
1GB of drivespace (10GB is a recommended minimum if running jenkins as a Docker container)

Recommended Hardware configuration for a small team:

4GB+ of RAM.
50GB+ of drivespace.

Software requirements:

Java 8/11 runtime environment are supported in both 32-bit and 64-bit version.

→ Goto Ec2 Instance. → Launch Instances → Jenkins
Select Redhat linux → Keypair → SSH port → Launch.

→ Add port : 8080 → Security group → Inbound →
Custom Tcp → 8080 (Tomcat) → Anywhere.
Save rules. (Jenkis required Tomcat)

→ Connect Instance → ec2-user,

→ Sudo -i

Step1: hostname Jenkis

vi /etc/hostname : Jenkins : wq!

bash

yum update -y

yum install vim wget net-tools -y

Step2: Install Java.

yum install java-11 -y

Java -version

Step3: Installing Jenkins

Google → Jenkins Installation (1st link) → Linux (Select)

Fedora (Redhat)

1. wget - - - - [To set repository]
2. rpm - - - - [Setting Jenkins licence key]
3. dnf-update.
4. yum install Jenkins -y
5. Systemctl daemon-reload

   Systemctl start Jenkins.

   Systemctl enable Jenkins.

   " Status " "

→ netstat -pantl

p = proces
a = all
n = numerice
t = tcp.
l = list

process

→ public IPv4 (copy) → Go to web browser IPv4:8080

→ Authentication : [unlock]

↓

copy the url [location].

↓

Jenkins server: cat url /var/lib/jenkins/secrets/

initialAdminpassword :

↓ passwd: ———— // [copy]

→ paste to unlock //

→ click on Suggest plugin Installation.

!! Creat a admin user.

username : Ravinadh

password : — — → Ravi

confirmation : — —

Full name : Ravinadh

E-mail id : ravinath8832@gmail.com → Save & continue

Jenkins Url →

↓ → Save & finish //

Start using Jenkins //

↓ configureation

→ /var/lib/jenkins → default location of Jenkis file

* project1: Creating Shell project:

Goto Jenkins dashboard → click on NewItem →

(myfirstshell project) → Typeofstyple: (Freestyle project)

↓ ok

config

Description: my shell script 1st project.

→ Source Code mangement: None

→

→ Build Steps → Add buildstep

↓

Execute Shell (Select)

```
#! bin/bash
echo "welcome to Jenkins"
echo "To Day Date is : `date`"
                ↓
echo "The build is executed at : `date`"
```

↓

Save

→ Click on Build Now ——→ [ ] ——↑——→ console output

→ Incase build is failed, to modify the project:

under my 1st shell projects → Configure.

* project2: Create a Shell project print 1 to 10 number:

NewItem → Second projects → Freestyle project → save

mysrcondshell project → buildstep → Execute shell →

```
#! /bin/bash
for i in 1 2 3 4 5 6 7 8 9 10.
do
if [ $i == 5 ];
then
sleep 30.
fi
echo "Iteration number $i"
done
```

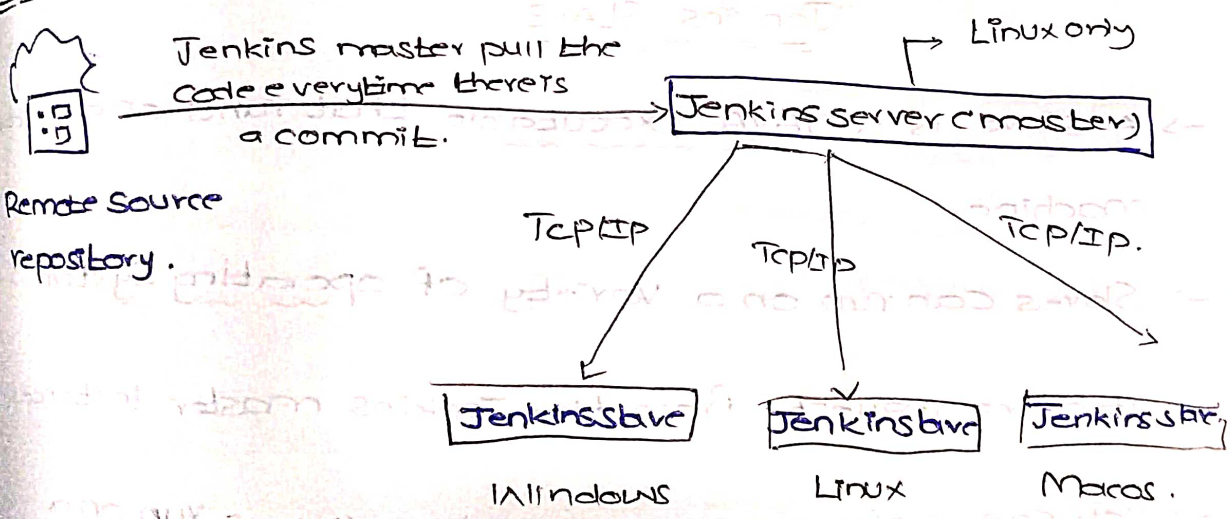
Realtime :-

30,000 server [Shutdown → 3 days holidate] → based on particular time. [Cortab]

→ Deleter project

Jenkins master-slave Aritctecture.

10/2/23



→ Jenkins master distrubutes the workloads to all slaves

→ one request from Jenkins master, the slave carry out builds and tests and produce test reports,

*. Jenkins master:-

→ The Jenkins master simply represents the base installation of Jenkins

→ The master will continue to perform basic operations server the user interface, while the slave do the heavy lifting.

→ master will be scheduling build Jobs.

→ Dispatching builds to the slaves for actual execution

→ monitor the slaves [Health checks]

→ Record and presenting the build results.

→ A master instance of Jenkins can also execute build Jobs directly.

Jenkins SLAVE:

→ A slave is a JAVA executable that runs on a remote machine.

→ Slaves can run on a variety of operating System.

→ It hears request from the Jenkins master instances

→ you can configure a project, to always run on a particular slave machine or simply let Jenkins pick the next available slave.

Step1: Launch Instances → Amazon linux → master.
Slave1
Slave2 "

→ hostname Slave

→ vi /etc/hostname : Slave : wq!

bash

yum update -y

yum install vim net-tools wget -y

yum install java-11 -y → java -version.

→

Step2: To communicate with master and Slave

1. change ssh setting :-

vim /etc/ssh/sshd_config

uncomment → permitRootLogin yes.

→ publicAuthentication yes.

passwordAuthentication yes : wq!

→ Setting password for root user:

passwd root → 1 [set passwd]

Systemctl restart sshd

→ Create a director :- mkdir /opt/ build-projects
Jobs

↓

Go to Jenkins dashboard, Ipv4 + 8080:

↓

private

Step3 :- Adding Slave node :

Under manage Jenkins → manage Node and 'cloud' →

+Newnode (nodename = slave1) → Type = permenantagent

Create //

Name : Slave1

Description :

Number of execution : 2.

↓ parallely how many Jobs to execute.

Remote : → /opt/build-Jobs.

Labels → Slave1

Usage → use this node as much as possible → both master and Slave execution both.

Launch method → via SSH

Slave : (privateIpaddress) → Host : [ ]

{ Credentials → Jenkins :

↓

username → root.

passwd → ----

ID → NODE1

Desc → NODE1 → Add }

Host key verify → Non-verifying Strategy :

Availability → Keep these node, online as much as possible.

Node pr → Nill

→ Add

verify → log

creating project:

New Item → myproject1 → Description → Label (preview)

restrict where this run (slave1) → Build steps → Execute shell

command: #!/bin/bash.

echo "Slave1 hostname: "hostname" " → Apply

→ cd /opt/build-Jobs

ls → remoting.jar [automatically remote]

cd workspace/.

ls.

2. Ec2 → linux → launch. [slave2].

#!bin/bash.

yum update -y

yum install vim wget net-tools -y → connect

→ hostname slave2.

vi /etc/hostname : slave2 : wq!

bash.

→ Go to Jenkins dashboard → manage nodes and cloud.

Name: slave2.

slave2.

mkdir /opt/buildjobs [remote location.

→ Launch

2nd method:- Slave connect to master by using controller.

Launch method :- Controller. → use websocket ☑ → Save

→ Click on save2 :- (Run on slave2) → 3 commands,

↓

① curl -so http://... -jar [agent.jar] fit, download

• agent.jar . [copy → wget ———] → enter,

(linkaddress,

wget

② → copy and Execute. → current Ipaddress not working.

change Ipaddress. [Ipaddress detail],

13/2/23:

Apache Maven · [build → to create executable file]

→ It is an open source software project management and comprehension tool

→ maven can manage a project's build, reporting and documentation from a central piece of information.

maven deals with several area of concern:

→ making the build process easy.

→ providing a uniform build system.

→ providing quality project information

→ Encouraging better development practise

Build tools.

→ Build tools help us to create an executable application from the source code.

→ The build tool is needed for the following processes.

1. Generating Source code.
2. Generating documentation from the source code.
3. Compiling Source code.
4. packaging the compiled codes in to JAR Files
5. Installing the packaged code in the local repository, server (or) central repository.

Stages of Build.

Src → Compile → packageit → Archieve → Deploy to Server/ Jvm

Installing maven :-

Launch Instance → maven → AmazonLinux → 22 → Launch

→ hostname maven.

vi /etc/hostname. → maven : wq!

bash.

yum update -y

yum install vim net-tools wget -y

Installing Java :-

yum install Java-11 -y

Java -version

Note: Build artifacts, those are like war and Jar file, Java executable files.

* Downloading and Installing maven :-

```
cd /opt
ls.
```

Google maven download -> Apache M D -> binary [copy link]

```
wget "link"
ls
tar -xvf filename.
mv apache-maven-3.9.0 maven3.
rm filename.
ls.
cd maven3/
cd bin/ → we have a software maven [mvn]
```

Setting mavenpath :-

```
echo $PATH
export PATH = /opt/maven3/bin:$PATH
echo $PATH
mvn --version
```

persistant setup.

```
vim ~/.bashrc.
export PATH = /opt/maven3/bin:$PATH
:wq!
```

Configuration file → Saved path

updated a file: source ~/.bashrc → [update bashrc file]

maven --version.

* MAVEN ARCHETYPE *

- Archetype is a Maven project templating toolkit.
- Archetype is defined as an original pattern or model from which all the other things of the same kind are made.

Note:- https://maven.apache.org/archetypes/

* Maven Core Concepts (pom File):

maven is created around the concept of pom file (project object model)

→ A pom file is an xml representation of project resources like source code, test code, dependencies (external JARs used) etc.

→ It contain a detailed description of your project, including versioning and configuration managment, dependicies, application and testing resources, team members, structure and many more.

→ The pom file should be located in the "root directory" of the project it belong to

pom.xml File Attributes:

Before maven 2, it was named as "project.xml" but since maven2, it was renamed as "pom.xml".

project: The root element of pox-xml file

GroupID :- Subelement of project - It specified the id for the project Group.

ArtifactId: The subelement of the project. It specifies the id for the artifact (project) - An artifact is (executable filename)
something that is either produced (or) used by a project
Example of artifacts : JARS, Source and binary distribution and WARS

version: The subelement of project. It specifies the version of the artifact under given group.

Imp [Maven Build Lifecycle] *

*. Generating a project :-

To create a simple Java Web application :-

Step1.

mvn archetype : generate -DgroupId = com.sysgeek.app -DartifactId = maven-app -DarchetypeArtifactId = maven-archetype-quickstart -DinteractiveMode = false

Step2. Go to cd /root/maven-app/

ls → src pom.xml

vim pom.xml

↳ verify details

:wq!

maven Build lifecycle

→ maven is based around the central concept of a build lifecycle. What this means: building and distributing a particular artifact (project) is clearly defined.

→ There are three built-in-build lifecycles:

1. clean:- The clean life cycle handles project cleaning
2. defaults: The default lifecycle handles your project deployment. Compile. → (Actions)
   Test Compile.
   Test
   package.
3. Site: The site lifecycle handles the creation of your project web site.

→ mvn clean. [clean the current project]

→ mvn package

→ ls → [target file created] Src pom.xml target [folder]

→ ls target [it contain .jar file]

* vim pom.xml
  ↓

<version> 15.0 tomcation. </version>

<name> tomcat </name> : 1212 !

mvn package.

ls "artifactId"

ls target/ →

14/2/23.

Git and Maven Integration:

→ Starts Jenkins and master machine.

Yum install git -y

git --version.

→ Go to github → create a Repository → and make a file with name script.sh

Code. Iteration of numbers.

→ In Jenkins → manage Jenkins → manage plugins →

Available plugins 1. (Git plugin)
2. Git hub integration plugin } Insta

→ Under New Item → Git-project → Freestyle project → ok

Source code managment → Git

[Code → url Copy] → paste

Repository URL = copy url

Branch to build = */main [Source code under Which branch]

Build Triggers = poll scm

Schedule. * * * * *

Build steps: Execute shell

↓

Command: bash script.sh

↓ Apply

Build Now

configure tab → General → Restrict where the project to be run. [Slave1] Select,

cloning the git reporistory,,

↓ Jenkins permission need to set,

Maven Integration :-

First Install maven on Jenkins Server

mvn --version.

cd /opt/maven3., pwd.

Go to manage Jenkins → manage plugins → Installed

Maven Integration plugins [download now and install after restart]

manage Jenkins → Global Tool configuration → Add paths

Add JDK → Name: my-Java.

JAva-Home path : /usr/lib/Jvm/ java-11-open Jdk...

→ find / -name java-11* [find path]. Slave1 to

path to Git executable :- Where is git.

/usr/bin/git

Maven → Add maven →

Name: maven.

uncheck.

maven Home :- /opt/ maven3

↓

Apply & Save

* NewItem → maven build → maven project / Freestyle → ok.

General → Restrict (slave1) → SCM → Git → Repo Url

pom.xml [uri]

Branchname : */main

→ pre steps
↳ Add pre-build steps → Invoke top level maven target.

Goals: clean package.

Rootpom → pom.xml

↓

Apply & save

* password for Jenkins

vi ~~/etc~~/var/lib/Jenkins/config.xml

↓

7th line Security → false ,,

Jenkins pipeline

16/9/23

→ It is also called as PAC (process as a code)

→ A pipeline is a collection of events or Jobs which are interlinked with one another in a sequence.

→ A pipeline has an extensible automation server for creating simple (or) even complex, delievary pipelines "as code", via DSL (Domain-Specific language).

Jenkins File

→ Jenkins pipeline is written into a textfile (called Jenkins File)

→ Jenkins file can be defined by using either webUI or with a Jenkins file.

The benifits of using Jenkins file:

→ make pipeline automatically for all branches, and can execute pull requests with just one JenkinsFile.

→ You can review your code on pipeline

→ This is the singular source for your pipeline and can be customized by multi-users.

pipeline Syntax :- [2 types]

1. Declarative :- It offers a simple way to create pipeline It consists of a predefined hierachy to create Jenkins pipeline.

→ It provides you the ability to control all aspects of

a pipeline execution in a simple, straightforward manner.

Scripted:- It runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to convert the pipeline in to atomic commands.

Note:- Both Scripted and declarative syntax are different from each other and are defined totally different.

Syntax

Declarative pipeline

```
pipeline {
    agent any
    stages {
        stage ("build") {
            steps {
            }
        }
    }
}
```

Scripted pipeline

```
node {
    stage () {
        // groovy script
    }
}
```

Example:- **Declarative Pipeline :-**

→ Dashboard → manage Jenkins → manage plugins → under available plugins → pipeline (Download and restart)

→ Create a NewItem → MyPipeline (name) → pipeline → OK

configure :→ pipeline → 2 method → Defination → pipeline script

```
pipeline {
agent Node any
    stages {
        stage (message) {
            steps {
            echo "Welcome to pipeline..."
        }
    }
}
}
        stage ( BUILD)
        {
            steps {
            echo "this is Build stage..."
                }
            }

        stage ( Test)
        {
            steps {
            echo "This is Test stage..."
                }
        }

        stage ( Deploy)
        {
            steps {
            echo "This is deploy stage"
                }
        }
```

* Ex-2 · NewItem → pipeline-scm → pipeline → ok.

Under pipelines → Definition → Select pipeline script

(Past Sample code)

→ In order to write url, use 'pipeline syntax' [Generate scripted syntax]

Steps Sample step → git : Git.

Repository url → https.

Branch → main.

↓

Generate pipeline script. → copy and paste

Stages (BUILD)

{

Steps {

Generate from pipeline script → git branch: 'main', url: 'https://github.com/Sysgeeksolu/piplegit'

}

}

Ex:3 · [2nd method]

NewItem → scm-pipeline → pipeline → ok.

→ Under pipeline → definitation → Select pipeline script from scm → sourcode → Git.

→ Goto Github → create a Repository (Jenkins pipeline)

↓

public

↓

Create Repo

↳ caps

→ Create a newfile (Jenkinsfile) in main

↓

write a pipeline code.

```
pipeline {
    agent any
    stages {
        stage ( BUILD )
        {
            steps {
                echo "Welcome to BUILD"
            }
        }
        Deploy
        Test //
    }
}
```

→ create.

Repo url: https://github.com/sysgeek4u/pipeline.git

Branch : */main

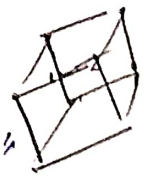
Scriptpath : Jenkinsfile → Apply save it → Build.

JIRA } → Ticketing tool
Nagios } → monitoring tool

Systems Engineer [Role]
↳ Linux
python
devops
AIMS

cloud Engineer

→ Welspark → bank [ ] purpose only



Samba

→ NFS