**Sonal Mittal’s Git Class**

1. **What is Git?**

Git  is free and open source software for distributed version control: tracking changes in any set of [files](https://en.wikipedia.org/wiki/Computer_file), usually used for coordinating work among programmers collaboratively developing source code  during  software development.

Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

1. **Installation of Git through AWS EC2 Instances**

* AWS EC2 ---> Create Instances ---> Select Amazon Linux Amazon Machine --->Select t2.micro instance type ---> select Key pair login & Create new RSA key pair and download the key ---> Network settings ---> Create security group ---> select type SSH ---> Add security group rule ---> select All traffic protocols(SSH & HTTPs) ---> Launch Instances ---> Local pc start ---> cmd --> cd downloads ---> click on ec2 instance ---> connect ---> ssh client ---> copy the ssh key ---> paste the same ssh key in local pc command prompt ---> Enter ---> sudo su -
* Install git(yum install git, git –version), Create Local repository(mkdir project, cd project, git init)
* Create file & Execute commands to add files to staging area and commit them to local repository
* Check git logs and Modify the files and commit existing files to local repository
* .gitignore workflow in git and Revert the commits from local repository

1. **What is the version control system?**

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are **software tools that help software teams manage changes to source code over time** (**at what time changes happen, who did the changes and collaborating the changes with other developers**).

**Various version control systems**: 1) Local, 2) Central, & 3) Distributed Version Control system

**Local + Central = Distributed version control system**

**Local Repository** ---> Maintaining a copy / version of your files in your local PC using a tool – Git.

**Remote Repository** ---> Maintaining a copy / version of your files on a remote server, where everyone can Access and Share their files using a tools like Git & GitHub.

1. **Git configuration to set up a username and email address? There are 3 types**
2. System : configuration of git done for entire system
3. Global : configuration of git done for that particular user you are logged into the server(root)
4. Local : configuration of git done for that particular folder (myproject) where git installed

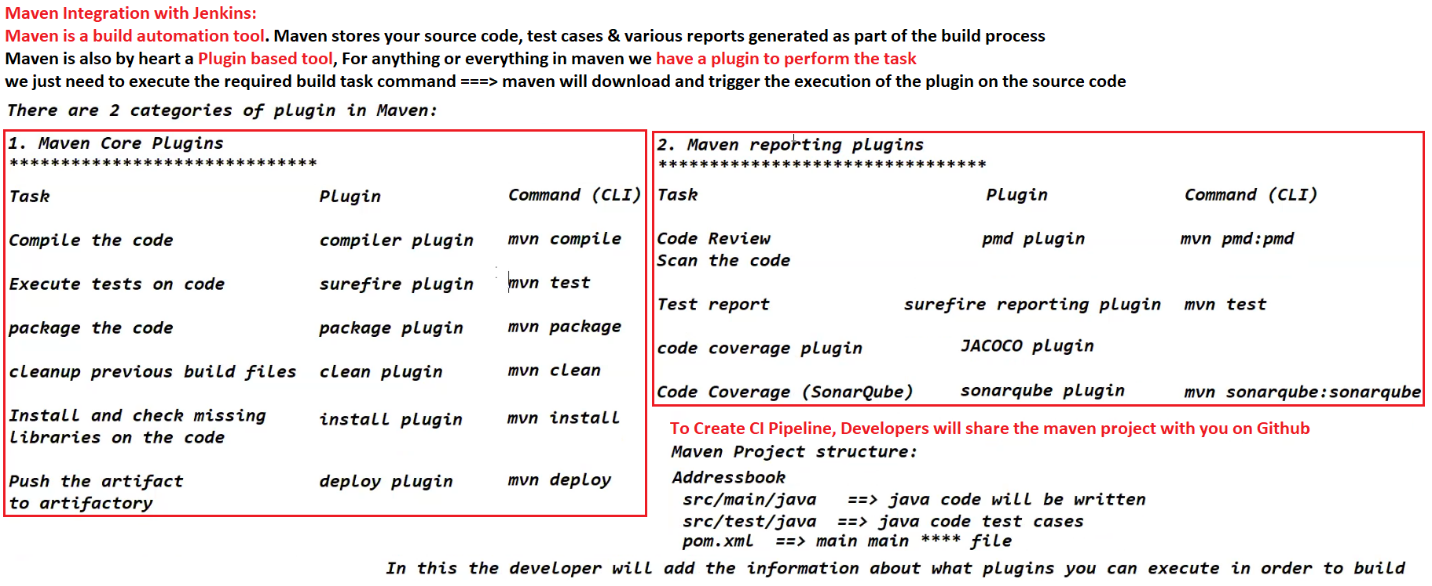
$ git config --global user.name VenkataNarasimha

$ git config --global user.email [venkatnarsimha@gmail.com](mailto:venkatnarsimha@gmail.com)

1. **Interview Question: List all the config done by you on your git repo**  ---> $ git config –list
2. **In which file &** **Where this Git config file saved? -** --global config will be saved at: ~/.gitconfig, $ cat ~/.gitconfig(Global Level), $ cat .git/config(Local) , $ cat etc/gitconfig(System Level).
3. **Add the file from working directory to staging area?**

$ git status ----- $ vi index1.html ---- $ git add index1.html ---- $ git status

1. **Commit the file from staging area to Local Repository?**  $ git commit -m "added file", $ git ls-files
2. **Git log:** $ git log and $ git log --oneline
3. **Git show**: $ git show or $ git show <git commit id>
4. **Interview Question: Save the commit history in a txt file?** $ git log >> /pathoffile/filename.txt
5. **Modify an exisitng file, which is already in Local repo** $ vi index1.html – added some data & to LR $ git add index1.html $ git commit -m "modification" (or) $ git commit -a -m "updated index1.html"
6. **Reverting the changes :** $ vim file3 (create a new file), $ git add file3, $ git commit -m "added file3" $ git rm file3, $ git commit -m "deleted file3", $ ls, $ git ls-files, $ git log --oneline // last commit id is of delete action, copy it -- lets revert the commit, $ git revert <commit id> -- file back to WD & LR
7. **To check difference between 2 version of a file?** $ git diff filename (or) $ git diff commit1 commit2
8. **.gitignore concept:** It is a concept where you can ask to git to ignore certain files from your working directory. In the WD, create few file: touch file.xml config.xml data.txt. -> git status, -> git will tell these files are untracked. But we don’t want these files to be version controlled. In this case, create a .gitignore file -> Open this file and add the names of all the files or directories save the file. And git will not track the files names & directories mentioned in .gitignore file. **Note**: This scenario is applicable to files that are with status as un-tacked.
9. **Deletion of file from WD & Local Repo?** $ git rm <filename>, Non-emptydir (sudo rm -r <filename>)
10. **Deletion of file local repo only?** git rm –cashed< filename>, git commit –m “deleted file from local”
11. **Git Reset:** This command is applied on your commit history. $ git reset <commit id>
    1. When we use the reset command, we are resetting the HEAD of the commits(last commit)
    2. This command is used when you want to discard changes done as part of multiple commits
    3. This command will also delete your commit ids, you will not be able to see the commit ids
    4. Once reset command is executed it never generates a commit id & log for reset operation
12. **Git Reset Types:** git reset –hard <commit id>, git reset –soft <commit id>, git reset –hard< cmt.id>
    1. Hard->will delete permanently from WD, staging area, LR
    2. Soft->will delete permanently from LR but the files will be available in WD and staging area
    3. Mixed->will delete permanently from LR and staging area but files will be available in WD
13. **Create a Remote repositories: github->New->public->create->Dev. Settings->Create Access Token**
14. **Connect Local repo to Remote Repo:** 
    1. $ git remote add origin <git hub repo URL> ->
    2. $ git remote –v -> Execute the command to check local and remote repo is set
    3. $ git remote set-url origin <url> ->It used to update existing remote origin
    4. $ git push origin master -> Pushing the file to remote repo from LR -> Enter the git hub username & password(Git Hub access token) **Note:** you will not be able to see the token text on screen.. Go ahead and press enter key
    5. $ git pull origin master -> Pull the files to your local repo from remote repo
    6. **Pulling a specific file form a remote repo to local:** Create 2 new files in github. In the local git CLI, $ git fetch origin master -> this will only fetch the new changes from the remote repo and commits & it will not merge with WD & LR. To check $ cd .git & $ cat FETCH\_HEAD
    7. $ git remote rm origin -> To remove local repo and remote repo connection
    8. $ git diff <commit id**>** This will show you, the files which are in remote repo & not in your LR
15. **Branching & Merging:** Branching is a concept in git, which allows to you freely work on multiple features at the same time without impacting the main master branch
    1. Whenever we commit on our repository, we can see a branch named master which gets updated automatically.
    2. Master branch is the default branch which get created upon initializing a repository
    3. You can create any number of branched, a branch is copy of master branch
    4. You can switch to a branch and work/develop on it. Once development or changes are complete on a branch , you can merge it to master branch
    5. You can also rename and delete a branch but you cannot delete the master branch.
    6. $ git branch -> To check how many branches are in Local repo
    7. $ git branch f1-> Create a new f1 branch in our Local repository
    8. $ git checkout f1 -> Switch to a particular f1 branch
    9. Add new file & commit on branch f1 $ touch file1 $ git add . $ git commit -m “added on f1”
    10. $ git log –oneline -> check the last commit, Switch back to master branch, & compare the both branches-> git diff --name-status master..f1 (or ) $ git diff master f1
    11. $ git checkout -b f2 -> To create a new branch as well as switch to new branch
16. **Merge:** Add new file & commit on the new branch Switch back to master, Merge the changes from f1 branch to master -> $ git checkout master , $ git merge f1 master -> check the commit history**.** Always switch to the branch where you want to merge the files from f1 branch. $ git merge <source branch> <destination branch> $ git merge f1 master -> Check the commit history on master
17. Merge a single file form Remote to local repo -> $ git checkout origin/master <filename> -> $ git checkout origin/master fetchfile1 -> $ git status ,file will be staged, $ git commit -m “message”
18. **Fork a Repo:** A fork is a copy of a Repository . Forking is copying the remote repo from one account to another account in github and forking allows you to freely experiment with changes without affecting the original project.
19. **Git Clone: Cloning of remote repo to Local machine** -> git clone filename or git clone origin master
20. **Change the name of the branch** $ git branch -M f1 dev
21. **Delete a branch**: $ git branch -D f1 -> Make sure , you are not on the branch that is to be deleted
22. **List of branches that have not been merged to master ->** $ git branch --no-merged
23. **List of branches that have been merged to master ->** $ git branch –merged
24. **Push the Branch to Remote repository:** Create a new branch $ git branch feature ->, switch to the branch $ git checkout feature -> add new changes and commit-> $ touch newfilename -> $ git add . -> $ git commit -m “doneon feature” -> $ git push origin feature
25. **Rebasing in GIT**: Rebasing is a merging strategy that is always applied on a feature branch & It is never applied on Master branch. In case rebase the commit history on your git repository is re-arranged on the feature branch. The commit history will integrated as a patch of all the commits of master branch as well as feature branch commits.
    1. Checkout to feature branch & create 2 new commits on the same file vi benchmarks.rb -> git commit -a -m "modified new file on fetaure branch" – 2 times
    2. Checkout to master & make commits. touch file.rb -> git add . -> git commit -m " cm on master"
    3. Switch to feature -> $ git checkout feature -> $ git rebase master --> we are able to see all the commits of master & feature branches commits in feature branch after rebase.
26. **Tags in GIT** : Light weight tags: In this method the tag is directly appended to latest Head commit id. $ git tag release1- > $ git tag –list -> To see all tags on the LR. $ git log –oneline --> To see the tag associated to commit id. $ git tag -a v1.1 -m “july release”<commit id>$ git show <tagname>
27. **Stashing in GIT**: Stashing is done in LR --> Which is applied only to the files that are tracked by git and have incomplete changes or modifications. $ git ls-files -> $ git status ->
28. Take a file that is already tracked by git -> Make some modifications to it -> $ git stash ->
29. Changes will be stashed in .git folder /refs/stash file -> $ git stash list->
30. Stash files using interactive mode -> $ git stash –p ->
31. See what is saved in the stash -> $ git show stash@{0}
32. Delete the stash and move the changes back to WD -> $ git stash pop stash@{0}
33. Keep the stash and move changes back to WD -> $ git stash apply stash@{0}
34. Password less authentication with GitHub: Git can communicate with GitHub over https or SSH.
35. When git is connected to github over https, you will use Username and personal access token to push files in to repo by authenticating the user.
36. When git is connect to github over SSH - ->on the local machine you have to generate SSH Key -> SSH private key/ SSH public key -> To generate SSH key -> $ ssh-keygen -> Press enter key 3 times -> the key will be generated & public key willbe in/root/.ssh/id\_rsa.pub. Get the public key content. $ cat /root/.ssh/id\_rsa.pub -> You will see the public key & copy the public key. GITHUB →Settings → leftside click on SSH & GPG keys → click on new SSh key -> Copy the key in next box -> Save the key
37. **Cherry-pick scenario:** git branch -> git checkout f2->ls->touch login ->git add . -> git commit -m "f2 login"-> touch simha-> git add . -> git commit -m "f2 simha"-> git log --oneline-> git checkout master -> git cherry-pick <66922bf f86a4c9> -> ls

****

**Jenkins**

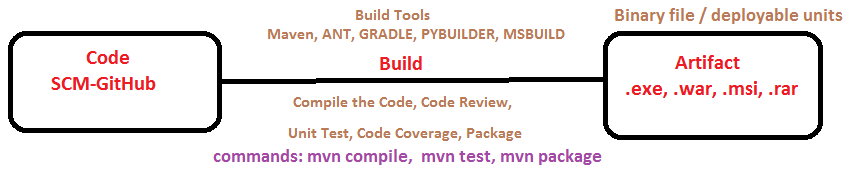
Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

Jenkins is a Continuous integration server which manages the control processes such as plan, code, build, test, deploy, operate, monitor in DevOps Environment.

Servers don’t understand your code, which are like python, java & rubee. It understands only 0s & 1s. It is our task to convert the code in to Artifact.

**What is Artifact?** Artifact is a package or deployable unit that can be deployable on server of various environments. Artifact is a package with an extention called .exe, .war, .msi, .rar files.

As DevOps Engineer, you need to automate the process of build. Build is a set of steps which are to be executed.



* **Compile the Code:** converting the code in to machine understandable language (0 & 1)
* **Code Review**: to check the programming mistakes, complexity of code, hard code data, vulnerable libraries.
* **Unit Test**: we will take few lines of code and run conditions to check the code is working or not ---> test is passed or failed.
* **Code Coverage**: Code coverage ---Tools, report ----> telling you how many lines of code has been tested and how many line of code has not been tested.

**Why Jenkins is so popular?**

* Open Source – Free tool & everyone can download and use. It is java based tool
* Good Plugin Support – Plugin is a Enhancement/Extension Feature to have better services
* Good Community support – it has good community to resolve the issues
* Fat and Reliable - It has GUI & Dashboards and Detailed console output
* Good OS Support - We can install & use in Windows, Linux, Ubuntu, RedHat, CentOS
* Scripted Builds – Pipeline as code

**Features of Jenkins:**

* Easy Installation Process
* Upgrades are easily available
* Provides advanced security
* Light weight container support
* Optimized performance
* Distributed Team Management

**Jenkins Installation Setup:**

* Create an ec2 Instance or use the same instance used in GIT sessions.
* Become root user ----->$ sudo su –
* If git is not present, Install Git ----->$ yum install git –y -----> $ git –version
* Install Java -----> $ sudo amazon-linux-extras install java-openjdk11
* On the browser open <https://www.jenkins.io/> ----> downloads section ----> Select Centos OS
* Go to Ec2 machine CLI and execute below commands to download the SW and Installation
* sudo wget -O /etc/yum.repos.d/jenkins.repo <https://pkg.jenkins.io/redhat-stable/jenkins.repo>
* sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io.key>
* Jenkins Installation ----> $ yum install jenkins –y -->To start service --> $ systemctl start Jenkins
* $ systemctl status Jenkins ---->To check Jenkins status. We can see, Jenkins enabled and running
* To set up Jenkins dashboard ----> copy public ip address of ec2 server ----> paste the IP address in web browser with Jenkins default port number. i.e. publicipaddress:8080.
* To unlock the pass word --->EC2 CLI --->$ cat /var/lib/jenkins/secrets/initialAdminPassword
* Copy the password and paste in the browser (jenkins) ----> Click on Continue
* Click on Jenkins suggested plugin tab(on left side) ----> Jenkins starts installing plugins
* Create username, password & update mail id--> Click on continue -->Click on finish --> Its Ready.

**How to integrate Jenkins with Github repository**

* Create a Clone Repo job in Jenkins to clone a GitHub repo in Jenkins workspace
* On the project click on Source code management ---> Select git option ---> Give git hub repo path & Branch name as Master ---->Save the job ----> Click on Build now ----> Cloned the Repo
* You can go to job → Jenkins workspace folder ----> able to see the cloned files.

**Jenkins Triggers:** To automatically execute a Jenkins job, we will use Jenkins Triggers.

**Periodic time- schedule:** Whenever there is a commit in GitHub repo, a parent job can trigger the execution of child job. Execute a job from remote script.

1. **Build Periodically**: Its automatically trigger the build as per the schedule irrespective of changes in repo. Go to Job Clone repo --> build trigger--> build periodically --> give \*/2 \* \* \* \* --> Save and build now.
2. **Poll SCM:** Build will be generated each time there is a change in the repository only. Build Triggers--> Poll SCM --> \* \* \* \* \*
3. **GitHub Hook trigger for GIT SCM polling**: Webhooks allow Jenkins to be notified when certain events happen on the repo. When the specified events happen, we’ll send a POST request to each of the URLs you provide. go to the git hub repository--> setting-->webhooks-->delete existing webhooks if any & --> click on create webhook--> Payload URL : Jenkins RL/GitHub-webhook/ ===> http://3.140.252.165:8080/github-webhook/Content type --> select application/JSON --> click on add webhook --> now make some changes in repo --> go to Jenkins--> you will see a new build has been created.

**Configure Build tools in Jenkins:** Jenkins--> Manage Jenkins-->Global Tool Configuration

1. **Java JDK**: Go to JDK section --> Uncheck -- install automatically --> JDK name as myjava -->Go to EC2 CLI server --> alternatives --config java -->Copy the path of java & add it in Jenkins for JAVA HOME field
2. **Git**: Under GIT==> leave it same as it is.
3. **Maven**: Under maven ==> lets install it automatically--> Provide the Name with version -->Save

**1. Code Compile:** Create a new Job Compile with freestyle project ---> Source code management --> select git--->give github path ---> <https://github.com/Sonal0409/DevOpsCodeDemo.git> --- > build--- > select Invoke top-level maven targets --> select maven version---> mymaven --> goal ==> compile---> save ---- > build now ----.> console output----.>O/P files /var/lib/jenkins/workspace/compile1/target/classes .

**2. Code Review:** Create a new Job CodeReview with freestyle project ---> Source code management --> select git--->give git hub path ---> <https://github.com/Sonal0409/DevOpsCodeDemo.git> --- > build--- > select Invoke top level maven targets --> maven version---> mymaven --> goal ==> pmd:pmd---> save - > build now ---.> console output---.>O/P files /var/lib/jenkins/workspace/target/pmd.xml file.

**Convert the file to Trend Report:** manage jenkins--> manage plugins--> available --> Search for warning next gen plugin--> install it.

Now go to the job **Code Review**==> go to post build actions ==> select recordcomplier warnings and static result analysis==>Under tool ==> select **PMD** ==> under report file format ==> give path of **pmd.xml** file ie: target/pmd.xml==> Save the file and build now ==> After the build is successfull, you will see PMD Warnings.

**3. Unit Test:** Create a new Job Code Review with freestyle project ---> Source code management --> select git--->give git hub path ---> <https://github.com/Sonal0409/DevOpsCodeDemo.git> --- > build--- > select Invoke top level maven targets --> maven version---> mymaven --> goal ==> test ---> save - > build now ---.> console output---.>O/P files /var/lib/jenkins /surefire-reports will be present.

Go back to job==> post build actions==> select junit test result job==>give target/surefire-reports/\*.xml

Here \*.xml is as we want to use all the xml files & you can see all pass and fail details of the tests.

**4. Code Coverage:** Create a new Job CodeReview with freestyle project ---> Source code management --> select git--->give git hub path ---> <https://github.com/Sonal0409/DevOpsCodeDemo.git> --- > build--- > select Invoke top level maven targets --> maven version---> mymaven --> goal ==> cobertura:cobertura -Dcobertura.report.format=xml---> save - > build now ---.> console output---.>O/P files /var/lib/jenkins/workspace/ target/site/cobertura/coverage.xml.

The xml file will be there, But we cannot understand them easily. So lets generate understandable reports by downloading cobertura plugin.

Go to Manage jenkins--> manage plugins-->Available→ cobertura --> install it ---> Coverage Report==> click on it ==> we cna see the coverage report.

**5. Package Job:** Create a new Job Complie with freestyle project ---> Source code management --> select git--->give github path ---> <https://github.com/Sonal0409/DevOpsCodeDemo.git> --- > build--- > select Invoke top level maven targets --> select maven version---> mymaven --> goal ==>package---> save → build now → console output → O/P files /var/lib/jenkins/workspace /target/ addressbook.war

**Connecting all the 5 jobs so that each job can be triggered**: Go to Manage plugins → available section, **build pipeline plugin** → click on Install without restart → Plugin will be installed.

**Pipeline as a code**: Let’s create a pipeline using code 🡪 Create a new job ==> select project as pipeline

pipeline {

tools {

maven ‘maven’

}

agent any

stages {

stage (‘Clone Repo’){

Steps{

git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'

}

}

stage (‘Code Compile’){

Steps{

sh ‘mvn compile’

}

}

stage (‘CodeReview’){

Steps{

sh ‘mvn pmd:pmd’

}

}

stage (‘Unit Testing’){

Steps{

sh ‘mvn test’

}

Post{

Success{

junit 'target/surefire-reports/\*.xml'

}

}

}

stage (‘Code Coverage’){

Steps{

sh ‘mvn cobertura:cobertura -Dcobertura.report.format=xml’

}

}

stage (‘Package’){

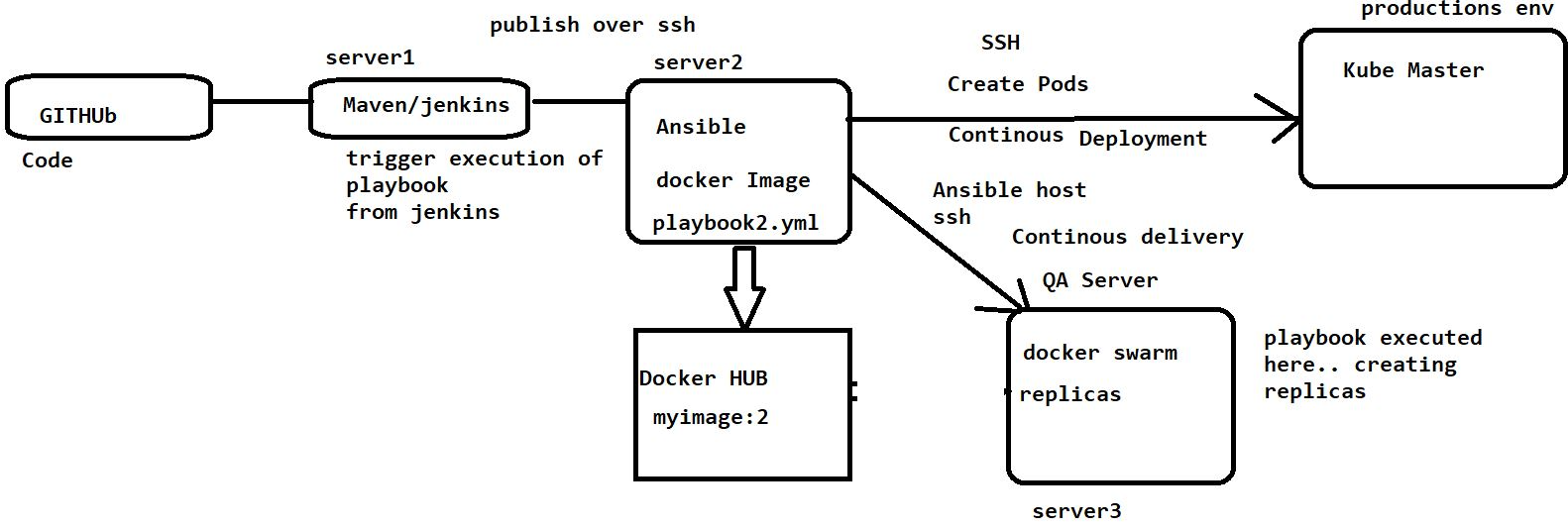
Steps{

sh ‘mvn package’

}

} }

**CICD PileLine**



Diagram

Description automatically generated with low confidence

**Pipeline as a Code**

In Jenkins to create a pipeline, code can be written in 2 ways. 1) Scripted 2) Declarative pipeline syntax

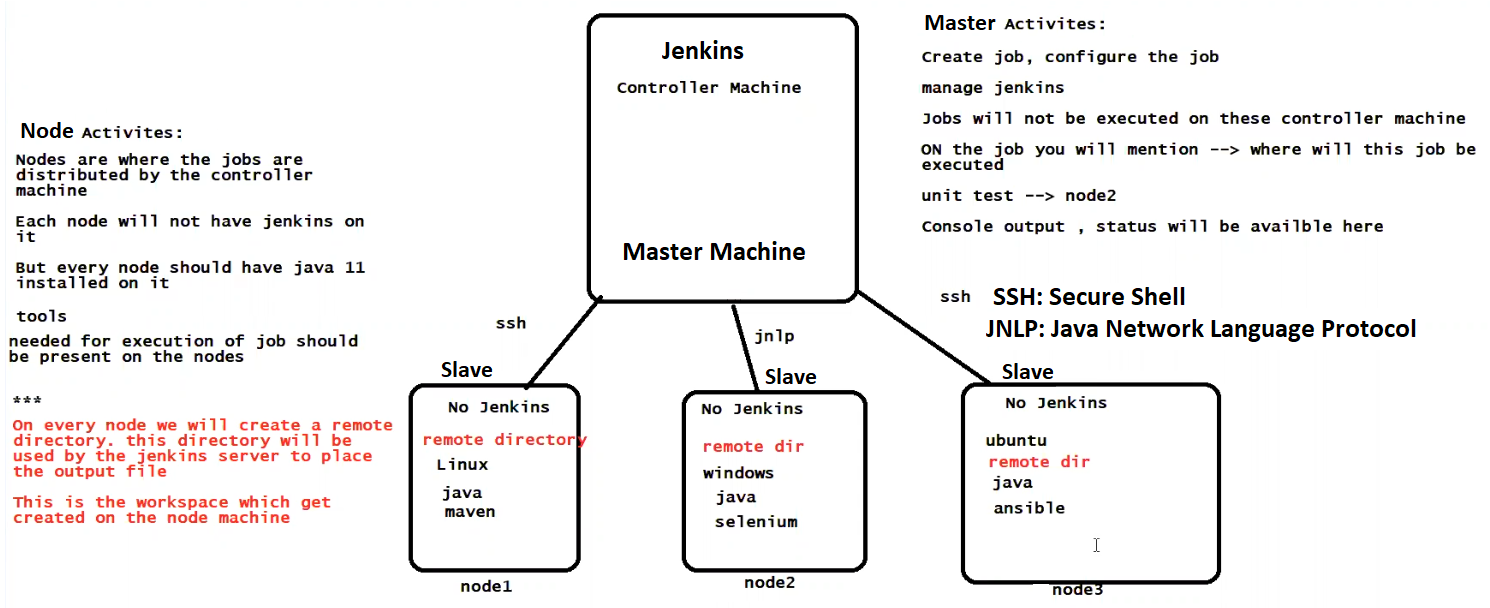
**Scripted pipeline syntax**: It is like DSL. It was very difficult to write and understand. there is no structure to the script and the developer can write the script in his own way. whenever the code is executed and if any errors occur, you will have to execute the code from starting and never validate the code.

it will start with Node.

**Declarative pipeline syntax**: It is easy to write code and the syntax will always start with a keyword pipeline but opening & closing brackets have to verify. the entire pipeline will have a single workspace.

It provides a structure to write the various tasks in the pipeline and we can include tools, agents, tasks, parameters, parallel tasks, and post-build steps.

Jenkins will validate the syntax errors. In case pipeline fails at any step, you can re-run the pipeline from that very step. The format of writing code is declarative (key 'value', name 'Simha', training 'devops')

****

**Jenkins Master**: Jenkins Master performs basic installation and handles tasks related to builds and configuration. It schedules build, Monitors slaves, Records & presents the build results. It will distribute the workload to the slaves.

**Jenkins Slave**: Slaves are basically set up to offload builds from the master and distribute the workload. It listens to the master's request. slaves can run on a variety of operating systems. slaves mainly execute build jobs which are dispatched by Jenkins Master instance.

**Master & Slave Node Setup:**

Setup the Slave Node, Setup the Master, connect them both over SSH, and distribute the jobs on the node from the Jenkins master

**Setup the Slave Node**: New Machine - Create Amazon Linux EC2 instance

Install java & create a remote directory (cd /tmp, mkdir jenkinsdir) and give read write permissions to the directory (chmod -R 777 /tmp/jenkinsdir). Make a note of the slave working or storage root directory path (pwd, /tmp/jenkinsdir).

**Setup the Master:**

Update the latest Jenkins URL in the configure system in the Master machine (Manage Jenkins ---> Configure System ----> Scroll down to jenkins URL & give latest URL)

Open the ports for communication between master & Node(agent) Manage jenkins ----> Configure Global security ---> scroll down to agents & select Random

**Connect them both over SSH:**

Manage Jenkins ---> Manage Nodes & Clouds ---> New Node ---> Give Node Name ---> Type-Permanent Agent ----> Name (Linux Node)----> Number of executors(1)----> Remote root directory( /tmp/jenkinsdir)---> Labels(Linux\_Node) ---> Usage(Only build jobs with label expressions matching this node) ---> Launch Method(Launch agent via SSH) ---> Host(Slave or Agents IP address) ---> Credentials-->Add ---> Jenkins---> kind(SSH username with private key)---> ID(nodeid)---> Description(nodeid1)---> username(ec2-user)--->Private key---> enter directly(pem file access key of EC2 instance)--->Add--->Select the Credentials---->Host key verification strategy(Non verifying verification strategy)----> Save----> Launching and Agent Node has been added to the Master

**Docker: (“ BUILD, SHIP & RUN ANY SOFTWARE ANY WHERE”)**

Diagram

Description automatically generated

**Docker:** Docker is a tool designed to create, deploy, and run applications with ease by using containers. Docker allows developer packaging of an application with all the requirements such as libraries and other dependencies, shipping it all as one package. It ensures that your application works seamlessly in any environment; be it Development, Test, or Production.

**Docker Installation** (yum install docker -y)----> Start docker service(systemctl start docker) ----> Check Status(systemctl status docker) ----> hostname DOCKERHOST---> sudo su - ----> docker images --> docker info(docker ps -a) ---> Pull & download the ubuntu(docker pull ubuntu or docker pull ubuntu:18.04 ) ---> docker images---> ls----> pwd---> cd /var/lib/docker---> ls---> cd image---> ls---> cd overlay2---> ls---> cd ---> docker images---> Delete the images in local host(docker rmi ubuntu:18.04 ubuntu) ---> Create a docker image & container and run (docker run –name cont1 ubuntu) ---> to check (docker ps -a) ---> to check docker memory info(docker stats) ---> To delete all the containers(docker rm -f $(docker ps -aq)) --->To delete specified containers(docker rm -f cont1 cont2) ---> Login to the Container or attach to container (docker attach <container id>)---> To come out of the container and keep it running(ctrl p&q )---> Stop running & come out of the container (exit) --> To know complete info(docker inspect<cont id>) --->To start container (docker start <container id>) --> To stop container (docker stop<container id>) -->

**Docker Modes:** Foreground Mode & Detached Mode

**Foreground Mode:** In this mode, Docker Container will be created, the container is up & running and the user will be attached to the container (user will be in inside the container) --->

To Create a container: docker run –name cont2 -it ubuntu(it means i-interactive, t-terminal) ---> In this container we can create the files & update the container---> touch file1, apt-get update, to come out of the container and keep it running( ctrl p&q )---> Login to the Container or attach to container (docker attach <container id>)---> To come out of the container & to stop running container (exit) -->

**Detached Mode**: Docker Container will be created & it is up & running and the user will not attach to it means the container will run in background & it is denoted by **-d** ---> To Create a container (docker run –name appcont -d nginx) ---> we can see, it automatically comes out of the container ---> docker ps -a.

**Access the Application on the container**: There are 2 ways to access the application on the container.

1. From the Docker host itself 2) From the external world (outside of the host machine) by script

**From Docker Host**: if we want to execute a command on the docker container via docker host, docker exec is the command. Ex: docker exec cont1 uname & docker exec cont1 ls -al /tmp (detached mode).

Ex: docker exec -it appcont bash (attached mode)

**From the External World**: if we want to execute a command on the docker container remotely, instance ip address with default port number. Ex: docker run --name n1 -d -p 8282:80 nginx.

**Docker Volume:**

Volume is a Directory where we preserve the data. Volumes are 2 types, Named & Bind Mount Volumes.

**Named Volumes**: Data of the container preserved from the local docker area. Create volume in docker directory. Ex: docker volume create myvol. data of container will be available at/var/lib/docker/volumes/myvol/\_data. Created a container & map to your volume

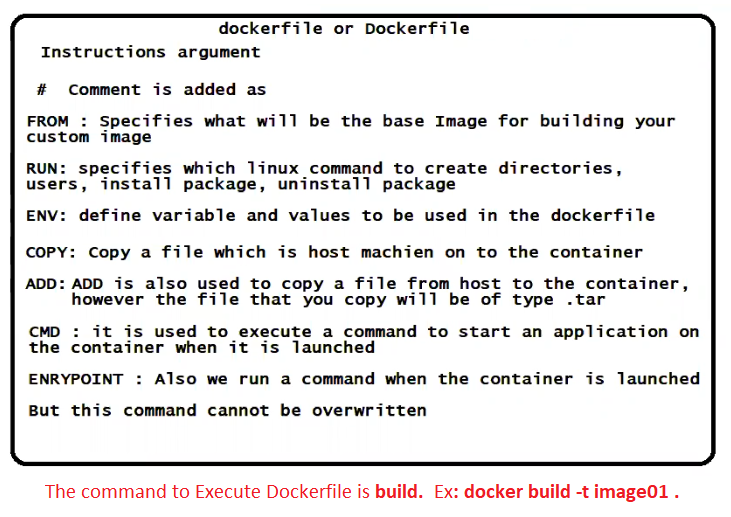
Please check once is the container data is preserved in the volume(dockerhost). Check if the dataform available on the container or not.

**Bind Mount Volume**: Bind Mount volume also can preserve the data of the container in the volume from any location or any directory of the host machine (Host Machine to Container Directory/volume).

**Dockerfile:**

A Docker file is a text file that is having a list of keywords given by docker and uses those keywords to form a docker file to create a docker image.

**Docker file**: It is just a text document that contains the instructions (values & Arguments) to build the image. The image is a read-only file and we can't edit. we need to run the image to create the container.



**Docker**: The main advantage of using docker is to run the applications in Isolation or Individually.

Ex: if you have two apps, 1st app needs to run on windows, and 2nd other needs to run on ubuntu in this case you won't be able to run these apps in a virtual machine because of the different OS in this case docker will be the best option to deploy your apps.

**Docker File**:

* Base OS should be ubuntu
* should be installed with 'nginx'
* Copy index file to /var/www/html/ of container
* Pass a command that will start nginx once container is created
* Expose container to port number(80)

vim index.html

<h1> this is code form Docker</h1>

<h1> Created by Narasimha</h1>

vim Dockerfile

#This is my first dockerfile

FROM ubuntu

RUN apt-get update

RUN apt-get install nginx -y

COPY index.html /var/www/html/

EXPOSE 80

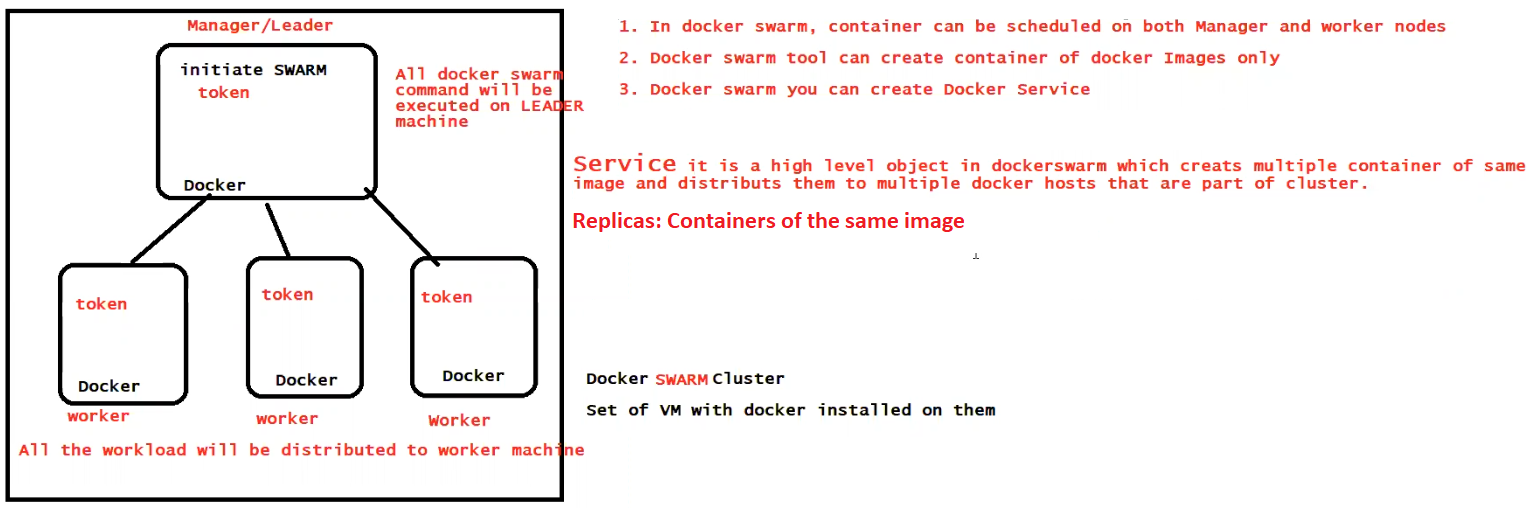
CMD [""nginx"", ""-g"", ""daemon off;""]

**Containerization**

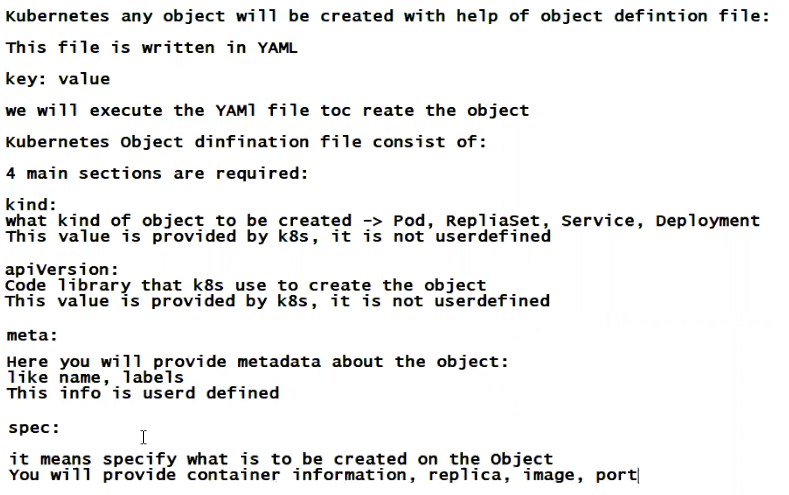
Diagram

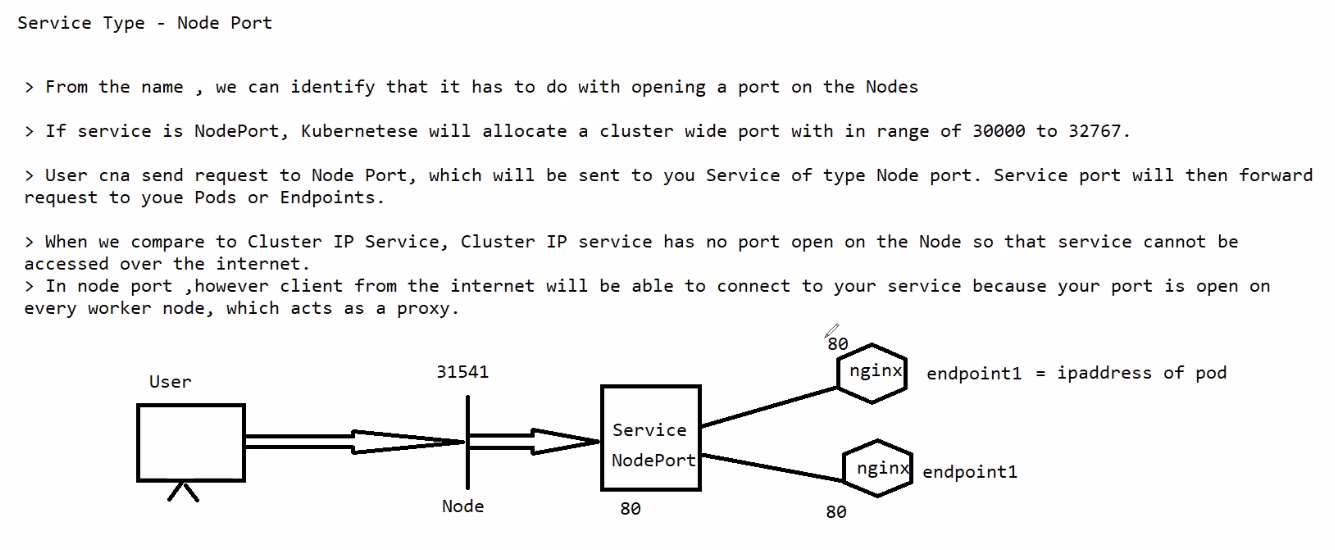
Description automatically generated

**Docker Swarm:**



**Kubernetes:**

****



**Ansible:**

