**GIT**

**git config --global user.name “Pradeep”** : to create user for git in your system

**git config --global user.email** [**pradeep@gmail.com**](mailto:pradeep@gmail.com) : to create user email git git in your system

**git --version : to check the version og git**

**git config --details: to check the git configuration details.**

**git init <repository>** : to create repository

**git init <repository>.git --bare** : to create bare repository

**git clone https://github.com/Pradipta-P/Novel.git <repository>** : to clone repository form remote repository

**touch 10.java** : to create a file named as 10.java

**git status** : to check the status of current repository

**git add <file>** : to staging the specified file

**git add .** : to staging the all files in repo

**git commit –m “message”** : to commit the changes

**git commit --amend** : to change the last commit message.

Note: if you change the message using --amend the commit id will also changed. So you have push the commit id to central repo using **git push --force** or **git push origin master --force**. Because already you have pushed the same changes so you need to forcefully push the new commit.

**git log** : to see the all commitID and its details

**git log --oneline** : to see the all commit in shot(oneline)

**git log --oneline –n** : to see the latest n number of commit in oneline

**git branch <branch\_name>** : to create a branch

**git branch** : to lsit all available branch

**git checkout <branch>** : to switch to the specified branch

**git merge <source\_branch> <destination\_branch>** : to merge data from source to destination branch

**git merge <source\_branch>** : to merge data from source branch to current branch

**git cherry-pick <commitId>** : to merge a specific commit

**git push --set-upstream origin <branch\_name>** : to make specified branch to a upstream branch in central repository

**git push –u –all** : to push everything from local repo to central repo including branches

**git pull <central repo> <branch\_name>** : to pull updates for central repo and particular branch.

Exa: **git pull** [**https://github.com/Pradipta-P/DevOps.git**](https://github.com/Pradipta-P/DevOps.git) **Development**

**git branch –d <branch\_name>** : to delete the branch

**git branch –D <branch\_name>** : to delete the unmerged branch

**git push origin --delete <branch\_name>** : to delete the branch from central repo

**git push --delete origin <branch\_name>** : to delete the branch from central repo

**git stash** : to create a backup of current state(uncommitted changes) and back to previous stage(commited)

**git stash list** : to list the all available stash

**git stash apply stash@{n}** : to go back to the previous stage(before the stash). n= Arry number 0=latest arry, highest number= oldest stash

**git stash clear** : to clear the stash

**git reset --soft** : to reset the change in repo(temp commit)(before the commit)

**git reset --mixed/ git reset** : to reset the changes in staging area & repository (before commit)

**git reset --hard** : to reset the change in all three satge(working dir, staging area & repository) before the commit.

**Git revert <previous commitid>** : to revert the change to previous commit(latestcommit) after the commit.

**Git checkout <commitid>** : to move to particular commit(moving the head pointer to specified commit)

**Git checkout <branch\_name>** : to move to the latest commit (moving the head pointer to latest commit)

**Git tag –a <parameter> -m <message> <commitid>** : to adda tag to particular commitid

Exa: git tag –a Relaese3.2.1 –m “1st release” 7a60003b

**Git tag** : to list the all tags

**Git show <tag>** : to see the details of tag(commitid)

**Git push --tags** : to push the tags to central repository.

**Git pull --tags** : to pull the tags from central repository

**Git tag –d <tag>** : to delete the specified tag

**Git push --delete origin <tag>** : to delete tag from central repository

**git push origin --delete <tag>:** to delete tag from central repository

Exa: git push --delete origin Release3.2.1

**Git fetch --all** : to fetcing all from central to local repo.(including branch)

**Git clean –n** : to see the all untracked file which are going to delete by **git clean** cmd

**Git clean –f** : forcefully delete all untracked files.

**.gitignore** : It’s a file basically used to ignore to push some specific files and folders to remote repo.

We have to mention the specific files and folders name inside the **.gitignore** (before staging)file so it will ignore to push the specific files and folders into central repo, even those files and folders are available inside the local repo.

**MAVEN**

**. mvn clean –pl <module name>**

**mvn version: set –DnewVersion= 2.3.0**

**-D**: defines a system property

-**P**: activate a profile

-**O**: works offline without accessing network

**- f** : specifies an alternate pom.xml file

**-s**: alternate settings.ml (path: maven homedirectory/conf/setting.xml)

**-fn**: never fail the build

**-q**: show only errors

**-X**: debug output

**Maven directory Structure** = E:\Lab\sonar\src\main\java\Esquare

**Unit test**: testing done by dev for particular unit or a small piece

**Integration testing**: This testing is used for different part of project works together.

**Smoke testing**: done by devops before handing over to QA for forther testing

**System testing**: This is also called as functional testing . where QA will check all functios are working or not.

**Regression testing**: This testing is done if any bug fix or code changes done. Test full product after modification.

**Performance testing**: testing the performance of product

**Beta testing**: testing done by user or customer

**Pom.xml**

<project>

<modelVersion> …. </modelVersion>

<groupId> …….. </groupId>

<artifactId> ……… <artifactId>

<version> …… </version>

<name> …… </name>

<properties>

…

…

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.8</version>

<executions>

<execution>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target>

<echo> Hello World</echo>

</target>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

**Jenkins**

Plugins:

1. GIt
2. Maven
3. SonarQube
4. [JaCoCo plugin](https://wiki.jenkins-ci.org/display/JENKINS/JaCoCo+Plugin)
5. Artifactory
6. [Build Failure Analyzer](http://wiki.jenkins-ci.org/display/JENKINS/Build+Failure+Analyzer)
7. [Build Pipeline Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin)
8. [Build Timeout](http://wiki.jenkins-ci.org/display/JENKINS/Build-timeout+Plugin)
9. [Email Extension Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin)
10. [Job Configuration History](https://wiki.jenkins.io/display/JENKINS/JobConfigHistory+Plugin)
11. [LDAP Plugin](https://wiki.jenkins-ci.org/display/JENKINS/LDAP+Plugin)
12. [Nested View Plugin](https://wiki.jenkins.io/display/JENKINS/Nested+View+Plugin)
13. [Pipeline](https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Plugin)
14. [Pre SCM BuildStep Plugin](https://wiki.jenkins-ci.org/display/JENKINS/pre-scm-buildstep)
15. [Publish Over SSH](http://wiki.jenkins-ci.org/display/JENKINS/Publish+Over+SSH+Plugin)
16. [SonarQube Scanner for Jenkins](http://redirect.sonarsource.com/plugins/jenkins.html)
17. [SSH Slaves](https://wiki.jenkins-ci.org/display/JENKINS/SSH+Slaves+plugin): Allows to launch agents over SSH
18. [ThinBackup](https://wiki.jenkins-ci.org/display/JENKINS/thinBackup)
19. [Throttle Concurrent Builds Plug-in](http://wiki.jenkins-ci.org/display/JENKINS/Throttle+Concurrent+Builds+Plugin) : This plugin allows for throttling the number of concurrent builds of a project running per node or globally
20. [Workspace Cleanup](http://wiki.jenkins-ci.org/display/JENKINS/Workspace+Cleanup+Plugin)

**Jenkinsfile**

pipeline {

agent any

stages {

stage ('Compile Stage') {

steps {

withMaven(maven : 'maven\_3\_5\_0') {

sh 'mvn clean compile'

}

}

}

stage ('Testing Stage') {

steps {

withMaven(maven : 'maven\_3\_5\_0') {

sh 'mvn test'

}

}

}

stage ('Deployment Stage') {

steps {

withMaven(maven : 'maven\_3\_5\_0') {

sh 'mvn deploy'

}}}}}

**Docker**

1. docker info : to check the docker information
2. docker run --name <c name> -d/-it –p <hp>:<cp> -v <hv>:<cv> <image> <cmd>
3. docker run --name <c name> -d --volumes-from <vs cont> --previleged=true <img> <cmd>
4. docker exec <c name/id> <cmd>
5. docker logs –f
6. docker ps –q : to list running container IDs
7. docker rm : to delete container
8. docker inspect <c name>
9. docker inspect --format ‘{{.NetworkSettings.IPAddress}}’ <c name/id>
10. docker top < c name/id> : to list the running process inside the container
11. docker stats <c name/id> <c name/id> : to show the live stream of container resource(CPU, RAM, ..) usage
12. docker events --filter event=attach --filter=die
13. docker events --since ‘1h’
14. docker images
15. docker pull <image name>
16. docker commit <c name/id> <myimage>
17. docker tag <image ID> <new name of image>

Dockerfile

FROM ubuntu

MAINTAINER pradeep

RUN apt-get update && apt install -y vim && apt install -y sudo

#RUN useradd pradeep && usermod -aG sudo pradeep

#CMD [" /bin/bash"]

#ENTRYPOINT ["echo", "Hello"]

WORKDIR /tmp/data

#USER pradeep

RUN sudo touch test

ENV NAME Pradeep

VOLUME ["/tmp/data"]

#COPY /root/data/file1 /tmp/data

ADD jdk-8u192-linux-x64.tar.gz /opt/

ENV JAVA\_HOME=/opt/jdk1.8.0\_192

ENV PATH "$PATH:/opt/jdk1.8.0\_192/bin"

EXPOSE 80

1. docker build -t <imagename> **.**
2. docker run –it –u <user name> -w <working directory> <image> <cmd>
3. docker build –t <image> -f ./<dockerfilename> **.**
4. docker login : to login central repo
5. docker login –u <user name> -p <password> <registry>
6. docker tag <localimage> <registry>/<repo>/<image>
7. docker push <registry>/<repo>/<image>

Exa:

FROM ubuntu

MAINTAINER Pradeep

RUN apt update $$ apt install –y apache2 && apt clean $$ rm –rf /var/lib/apt/lists/\*

ENV APACHE\_RUN\_USER www-data

ENV APACHE\_RUN\_GROUP www-data

ENV APACHE\_LOG\_DIR /var/log/apache2

EXPOSE 80

CMD [“/usr/sbin/apache2”, “-D”, “FOREGROUND”]

**Docker Swarm**

Open all these port before enabling swarm.

Ports: TCP= 22,2376,2377,7946 UDP= 7946, 4789

Exa: firewall-cmd --add-port=22/tcp

1. docker swarm init –advertise-addr <manager ID>
2. docker node ls
3. docker swarm join-token <manager/worker>
4. docker swarm leave –force
5. docker node promote <worker ID/ name>
6. docker node demote <worker ID/ name>
7. docker service create --name <name> --publish published=80,target=80 nginx
8. docker service ls
9. docker service ps
10. docker service scale <service name> = <number>
11. docker service scale mynginx=5
12. docker service update --image <image name> <service name>
13. docker service update --image httpd mynginx
14. docker node update --availability drain <node name>
15. docker node update --availability active <node name>
16. docker service create --name mynginx --mode global Nginx
17. to run service only in ubuntu system we can filter it using constraints(Exa: constraints == ubuntu)

**docker-compose.yml**

---

version: “3.4”

services:

<serviceName>:

Image: <image name>

stdin\_open: true # equals to –i (interactive mode)

tty: true # equals to -t

stop\_grace\_period: <number>s # number of seconds to wait before stoping the container

environment :

<varaiblename> : <value>

volumes:

* <host path>:<containerpath>

logging:

optinos:

max-size: <number>[m|k] # maximum size of log file(json)

max-file: <number> # maximum number of log files

ports:

-target: <containerport>

published: <Hostporrt>

protocol: [tcp|udp]

mode: [ingress|host] # ingress publish to load blance, hot publish on each node

healthcheck:

test:

* CMD
* <command to run HEalthcheck>

Interval : <duration> # default 30 seconds

timeout: <duration>

retries: <number> # number time to retry before considering unhealthy

start\_period: <number> # intial time to wait for start container after created.

Start healthcheck

deploy:

mode: [replicated|global]

replicas: <numbers>

update\_config: # for rolling update

parallelism : <number> # parallel how many container can update

delay : <duration> # time to wait between upating container

failure\_action: [roleback | continue|puse]

order: [start-first | stop-first]

restart\_policy:

condition: [on-failure|none|any]

delay: <duration> # how long to wait between restart attemps

max\_attempts: <number>

giving up

window: <duration> # how long to wait before deciding if a restart has succeded

**What’s the difference between up, run, and start?**

Typically, you want docker-compose up. Use up to start or restart all the services defined in a docker-compose.yml

The docker-compose run command is for running “one-off” or “ad-hoc” tasks

The docker-compose start command is useful only to restart containers that were previously created but were stopped.

**Can I use json instead of yaml for my Compose file?**

**Yes exa:** docker-compose -f docker-compose.json up

Docker rmi <imagename> : to delete the docker image

docker rmi $(docker images -q)  : to delete all docker images.

Docker stop $(docker ps –a –q): to stop all docker container

Docker rm $(docker ps –a –q) : to delete all docker container.

Docker Networks:

1. Bridge local
2. Docker\_gwbridge local
3. Host local
4. Ingress swarm
5. None locla

Ansible

---

- hosts: local

become: yes

vars:

download\_url: https://download.oracle.com/otn-pub/java/jdk/8u192-b12/750e1c8617c5452694857ad95c3ee230/jdk-8u192-linux-x64.tar.gz

download\_folder: /opt

java\_name: "{{download\_folder}}/jdk1.8.0\_192"

java\_archive: "{{download\_folder}}/jdk-8u192-linux-x64.tar.gz"

tasks:

- name: Checkin if Java is already installed

stat:

path: '{{ java\_name }}'

register: java\_installed

- name: Download Java

raw: "wget -q -O {{java\_archive}} --no-check-certificate --no-cookies --header 'Cookie: oraclelicense=accept-securebackup-cookie' {{download\_url}} creates={{java\_archive}}"

when: java\_installed.stat.exists != True

- name: Unpack archive

raw: "tar -zxf {{java\_archive}} -C {{download\_folder}} creates={{java\_name}}"

when: java\_installed.stat.exists != True

- name: Fix ownership

file: state=directory path={{java\_name}} owner=root group=root recurse=yes

when: java\_installed.stat.exists == True

- name: Make Java available for system

raw: 'alternatives --install "/usr/bin/java" "java" "{{java\_name}}/bin/java" 2000'

when: java\_installed.stat.exists == True

- name: Clean up

file: state=absent path={{java\_archive}}