**DevOps**

**Steps in DevOps:**

1. **Code Repositories(**Source Code Management)

Git, Github, Gitlab

SVN

TPS

1. **Build Tools**

(Clean, Package, Install, Unit test)

Maven (POM.xml)

Ant (build.xml)

Gradle (build.gradle)

1. **Continuous integration**

Jenkins

Bamboo

Team city

1. **Static code analyzer**

Sonar qube

1. **Integration**

(Regression testing)

Selenium

1. **Archive artifact**

Nexus

Artifactory

1. **Deploy**

Tomcat Server

**Log monitoring tools: ELK** (Elastic search, Logstash, Kibana), **Splunk**

**GitHub**

**Types of repositories:**

1. Public repository

2. Private repository

**Fork:**

To save a project from a public repository to our repository use the fork option. If it is a private repository, credentials are required.

**Branches:**

The project may have several branches based on updates or bug fixes.

**Clone command:**

To clone a project from github to the local directory, use the command:

**$git clone url**

**Checkout command:**

To switch between the branches in the project, use the command:

**$git checkout branchname**

**.git file:**

After cloning the project to the local repository, if we delete the .git file, the project cannot be built.

**Maven**

Maven is a build automation tool used for java projects. It needs to have the POM.xml file which contains the configuration details such as version, group id, artifact id, version of artifact.

The Source code of the project contains two files:

Source directory: src/main/java

Test directory: src/test/java

**To install maven:**

$ sudo apt-get install maven

**To check the version of maven:**

$ mvn –version

**$mvn clean package:**

It is the command to build and package the project. It does not create backup in the file directory.

**$mvn clean install:**

It is also to build and package the project. It creates a backup in the file directory.

**$mvn clean package –DskipTests=True:**

This command builds the project but skips the test script.

**Jenkins**

Jenkins is an open source software to automate the software development process with continuous integration and continuous delivery (CICD).

**Jenkins installation steps:**

<https://www.tutorialspoint.com/articles/how-to-install-jenkins-on-ubuntu>

After logging into Jenkins, go to “Manage Jenkins 🡪 Global tool configuration” and install any build tools necessary and specify the path for the software already installed.

**Building a new project:**

1. Go to “New item” and type the item name and select the type of project (Eg: Maven)
2. Under source code management select git and specify the url for the git repository and the branch to use.
3. Build Triggers can be set to the option that matches our need as to when the build has to take place.
4. The build goals has to specified such as “clean package sonar:sonar –Dsonar.host.url=http://ip –DskipTests=True /clean install”. If it is a freestyle project, the commands must be specified in the shell script of Build option.
5. After saving select the “Build now” option.

**Note:** If the project build success in prompt and failure in Jenkins or vice versa check the version matching of the plugins. If still a Failure check the code.

**Sonar qube**

Install and configure **Postgres** and **Sonarqube** following the steps:

<https://www.vultr.com/docs/how-to-install-sonarqube-on-ubuntu-16-04>

1. The code analysis results for projects built will be stored in sonar qube. When any project is built using sonar, the code analysis results are stored here.
2. It shows the reliability (bugs, vulnerabilities), security, maintainability, duplications, etc. If the quality criteria is not passed, the source code must be modified to meet the requirements.
3. The rules tab shows all the rules for each programming language according to which the source code is analyzed. The degree of impact, type of issue can be modified in this tab.
4. The quality profiles tab shows the various profiles that have been configured to perform the code analysis. A custom profile can be built with all the rules required for our project.
5. The quality gates tab is used to set the criteria for qualifying a project as passed/failed. Various conditions can be set here.

**Docker**

Docker is used to create a docker image and run it inside a container to provide an additional layer of abstraction.

To install docker use the command:

**$sudo apt-get install docker.io**

1. Create an account in hub.docker.com
2. To search for any image in docker hub:

**$docker search repositoryname**

1. To pull an image from docker hub:

**$docker pull imagename**

1. To pull a specific version(tag) of an image:

**$docker pull imagename:tag**

1. To display the images available in the file directory:

**$docker images**

1. To delete an image:

**$docker rmi imageid** (only when image is not used in a container)

**$docker rmi –f imageid** (to force remove)

1. To create an image from a file, create a directory and then create a file named “Dockerfile” and type the commands necessary. (<https://www.howtoforge.com/tutorial/how-to-create-docker-images-with-dockerfile/>)

**$vi Dockerfile**

1. Build the image from the file by executing this command inside the directory:

**$docker build –t imagename .**

Image cannot be edited. It can only be overridden to create a new image.

1. Create a container for the image

**$docker run –it imageid bash** (“it” means interactive mode)

**“exit”** command is used to exit the container

At this point, the image that was used to create the container can be forcefully removed.

1. To start an already existing container:

**$docker start containerid**

**$docker exec –it containerid bash**

In this state, even rmi –f cannot be used. We have to remove the container forcefully to stop it using the following command before removing the image.

After the image is deleted, the container cannot be started

**$docker stop containerid**

1. To delete a container:

**$docker rm containerid**

**$docker rm –f containerid**

1. To list the containers currently running:

**$docker ps**

1. To list all the containers:

**$docker ps –a**

1. To login to docker hub:

**$docker login**

Enter the username and password.

1. Create a repository in docker hub and an image to be uploaded to docker hub and then tag the image:

**$docker tag imageid username/repositoryname:tagname**

1. Push the image into the repository:

**$docker push username/repositoryname:tagname**

Tagname is optional. If it is not specified, the default tag will be “latest”.

**Nexus Repository**

**Intalling nexus:**

1. Download the nexus tar file

**$wget url**

1. Extract the tar file

**$tar –xvf filename**

1. Create a new user named nexus

**$adduser nexus**

1. Go to the file nexus/bin/nexus.rc and uncomment the line in the file and change it to **run\_as\_user=”nexus”**.
2. Open the file nexus/bin/nexus.vmoptions and change the path having “..” to the nexus path.
3. Create a symbolic link to the init using the command

**$ln –s nexuslocation /etc/init.d/nexus**

1. Change the ownership of the nexus file

**$sudo chown –R nexus:nexus nexuslocation**

1. Create a new session and then go into the nexus bin and change to nexus user using the command

**$su nexus**

1. Start the nexus

**$./nexus start**

**$./nexus run**

1. Go to localhost:8081 to access the nexus repository. Default sign in credentials is username: admin password: admin123