**CI/CD pipeline**

* all the maven process (or any other life cycle) can be automated in Jenkins
* continuously integrate the changes and continuously deploy
* AWS code pipeline is similar to Jenkins - CI/CD pipeline
* Git use yaml language
* Jenkins use pipeline language

Once the code is pushed to git 🡪 Jenkins will be triggered

Triggers - periodically, poll scm, webhook

Then it pushes to the Central Artifactory (Jfrog, for docker-docker hub etc.)

Then from Artifactory we pull the image and deploy it to the environment

**CI/CD Pipeline:**

**Git checkout**

* whether it should trigger every week or every month etc. we use periodically like crontab
* With trigger the jenkin life cycle starts.
* Poll SCM - will trigger life cycle only if there is any new commits or changes also like crontab
* Webhook - triggered simultaneously at that time itself without waiting periodically

**Build tools:**

* build tools - Make, Maven, Gradle

**Artifactory**

* we build package using build tools and push it to Artifactory - JFROG, dockerhub, ECR

**Deploying to Environment:**

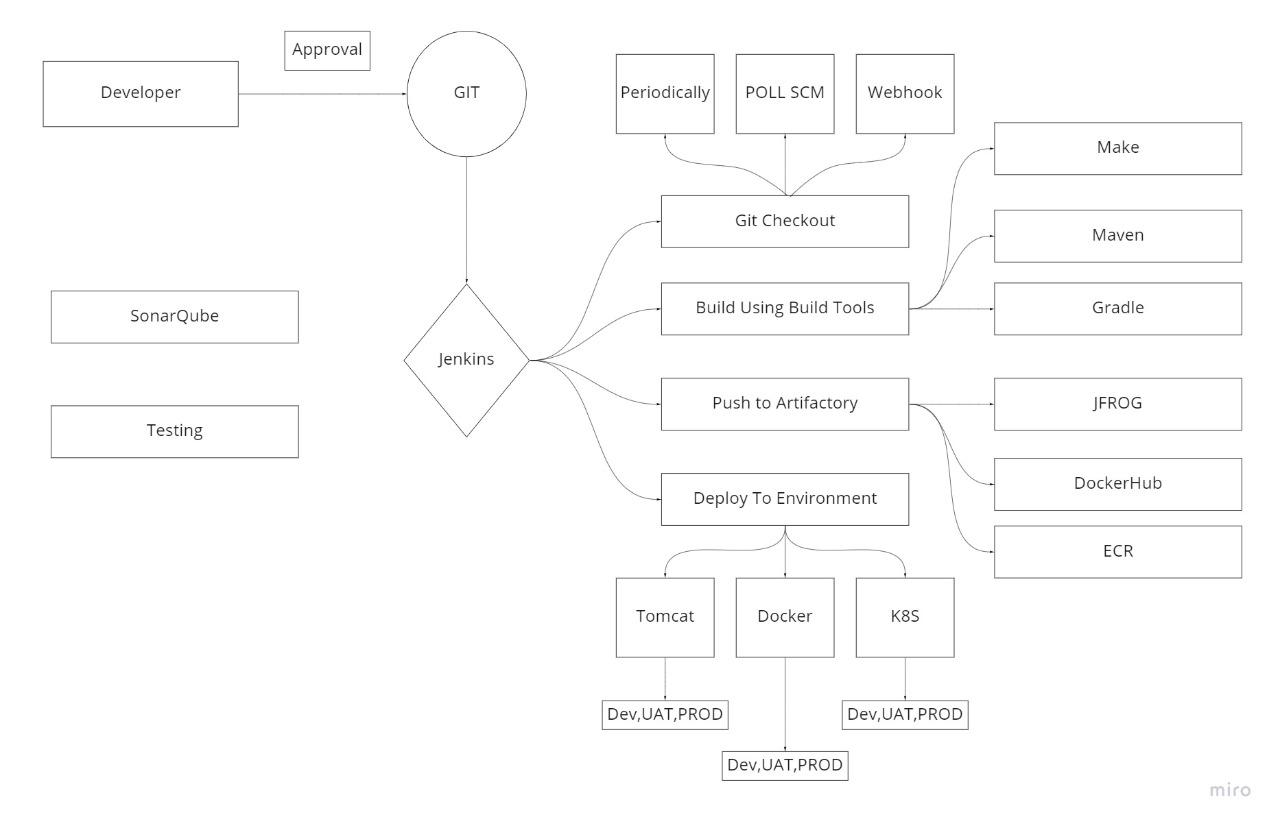
* then now we pull the package to the working environment (Dev/UAT/Prod) - tomcat/docker/kubernetes(k8s)

**SonarQube** - for code analysis whether code is good or not - 70% sonarqube analysis - mostly approved

* Sonarqube comes between developer and git for approval. Here Jenkins can automate it. this can also be integrated into Jenkins

**Testing**

* we can integrate testing cases in Jenkins life cycle



**Installing Jenkins in server**

1. setup instance - Ubuntu AMI – in aws
2. update packages
3. install java
4. download from jenkin.io 🡪 select ubuntu flavour
5. copy codes from Jenkins website
6. after installed, ipaddrees:8080
7. copy shown directory and get password using cat in server
8. install suggested plugins
9. install maven, git, make
10. log on to Jenkins

**Freestyle Job:**

Build Execute Shell

cd /var/lib/jenkins/workspace/<Job\_Name>

mvn clean install

**Triggers:**

**1. Build Periodically:**

* it is used to trigger the Jenkins job based on schedule (crontab) we have set

**2. POLL SCM:**

* Periodically (crontab) polls the scm (source code management) to check whether a new commit has been made. if there are new commits, it will trigger Jenkins job

**3. GitHub Webhook trigger**

* In Jenkins Webhook triggers are used to trigger the Jenkins jobs whenever a developer makes any changes to the repo

**Jenkins default path:**

/var/lib/jenkins

**Assignment**:

setup webhook trigger

**Master - slave server for Jenkins**

* We cannot install all services in just one server which may make it unresponsive
* main configuration and all 🡪 are done on main Jenkins server - master machine
* java builds, make builds, tomcat, docker, k8s 🡪 will be done on other servers called slave/agent machines.

Test environments may also have heavy loads

1. slave 1 - java project is uploaded to Artifactory (jfrog)
2. slave 2 - tomcat (dev) - webapps (.war)
3. slave 3 - UAT
4. slave 4 - prod environment

**Need for Master-Slave Concept:**

* In scenarios where you need several different environments to test your builds, which cannot be done with a single Jenkins server, then we can use a different machine/server with different environment as a slave to take the specific job from the master
* If the master server has limited resources, then we can perform operation on slave machine

**Master-Slave connection requirements:**

* Java Installed
* Password less connection between master and slave

**Advantage:**

* Master can be Ubuntu and slave can be red hat or any other flavours
* No. of Executors 🡪 How many different jobs that can run in parallel

**Steps to setup:**

* create a new server (red hat)
* log into Jenkins server
* manage Jenkins --> mange nodes and cloud --> new node --> we can create slave
* number of executors - how many different jobs that can run in parallel
* remote root directory: /home/ec2-user/Jenkins
* labels : slave1
* launch method : ssh
* host: pvt ip if same network, different network -
* credentials :
* manage Jenkins - manage credentials - system - global -add credentials - ssh private key id
* give private key
* key verification --> manual verification

In interview - we have 1 master and 7/9/11 slave server

**Remote root directory:**

* location on slave where Jenkins workspace is created
* default is 🡪 /var/lib/Jenkins

**Plugins in Jenkins**

* to integrate any tool with Jenkins (as initially it won’t provide except few)
* more than 1500 plugins in market

**Steps:**

* dashboard - in manage plugins - choose the one we need
* if we install monitor plugin - we can monitor server, load etc.
* blue ocean - plugin - new dashboard for pipelines

Assignment: lifecycle-sprints, agile sprint methods

**Plugins:**

* The plugins help in increasing the capabilities of Jenkins by adding some useful features to enhance productivity.

**Popular plugins:**

1. ~~CVS - Source code Management plugin (out of use - dont say in interview)~~
2. Monitoring Plugin - to monitor the jenkins master and its agents
3. docker plugins - to integrate docker with jenkins
4. kubernetes plugins - to integrate kubernetes with jenkins
5. periodic backup - to take backup of jenkins
6. blue ocean - modern personalized dashboard view
7. maven integration - to integrate maven with jenkins
8. mailer - to setup email in jenkins
9. Pipeline maven integration

**Matrix security**

Q: how do you secure Jenkins server?

- matrix based security or project-based matrix authorization strategy

A: manage Jenkins - configure global security - matrix based security or project-based matrix authorization strategy

* **Matrix authorization**: allows configuring the permissions for each user individually. If matrix based security is given - we need to give our own admin permission
* **Project based matrix authorization strategy**: an extension to matrix security where we can setup permissions to individual projects so that you can allow a specific user to access only certain projects or jobs.

**Name three security mechanisms Jenkins uses to authenticate users.**

Jenkins can authenticate users in one of three ways:

1. Jenkins can use an internal database to store user data and credentials. (This is the default.)
2. Jenkins can be configured to authenticate against a LDAP- Lightweight Directory Access Protocol server.
3. Jenkins can be configured to employ the authentication mechanism used by the application server upon which it is deployed.

**Describe the standard process to configure and use third-party tools within Jenkins?**

The process to use a third-party tool, such as [Artifactory](https://www.theserverside.com/video/How-to-set-up-and-use-a-JFrog-Artifactory-Maven-repository), Node, SonarQube or [Git](https://www.theserverside.com/tutorial/Five-basic-Git-commands-every-beginner-needs-to-know) typically follows a four-step process.

1. The third-party software must be installed.
2. A Jenkins plug-in that supports the third-party tool must be installed through Jenkins admin console.
3. The third-party tool must be configured in the Tools tab of the Manage Jenkins section of the admin console.
4. Finally, the plug-in can be used from within a Jenkins build job. The plug-in will then facilitate communication between the Jenkins build job and the third-party tool.

This is a tough Jenkins interview question for DevOps professionals because not every third-party tool is configured in exactly the same way. For example, Jenkins can be configured to [install Maven](https://www.theserverside.com/tutorial/How-to-install-Maven-and-build-apps-with-the-mvn-command-line) itself, rather than requiring a pre-existing installation. Similarly, third-party tools, like [Checkstyle](https://www.theserverside.com/video/Use-Maven-Checkstyle-Plugin-to-enforce-Java-quality-rules) or JaCoCo, can be downloaded at build time by Maven. So these four steps are not always adhered to strictly, but at a high level, these are the typical steps required to install and configure a third-party Jenkins tool.

**Declarative vs Scripted**

* groovy declarative (new way) - pipeline block
* groovy scripted (old) - script block

**CICD:**

Once dev team pushes the code to GitHub, a Jenkins pipeline job will get triggered. It has 4 stages

1. git checkout
2. build
3. push
4. deploy
5. In the 1st stage i.e., **git checkout**, Jenkins will pull the source code from git hub. once the stage one completes in success, build stage will get triggered automatically.
6. In stage 2 - i.e., **build stage**, Jenkins will compile the source code and generates a binary. Once build stage completes, push stage will get triggered.
7. In stage 3 - i.e., **push stage**, Jenkins will the push the generated binary to an Artifactory. After push stage, deploy stage gets triggered.
8. In stage 4 - i.e., **deploy stage**, the binary will be pulled from the Artifactory and gets deployed to the specific environment.

**Jenkins is a CI/CD tool**

In a company we use pipeline - using pipeline script

1. **Continuous Integration:** is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily. (develop - git -jobs)
2. **Continuous Delivery:** Continuous delivery picks up where continuous integration ends. CD automates the delivery of applications to selected infrastructure environments. Most teams work with multiple environments other than the production, such as development and testing environments, and CD ensures there is an automated way to push code changes to them. (Deploy changes - (war) test in different environments (dev, test, UAT etc.) then to customer)
3. **Continuous Deployment:** (directly deliver it to the customer) Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Q: what is your CI/CD setup?

Q: What is your branching strategy?

Example:

Tomcat

1. Stage 1 - pom.xml, dependency git checkout
2. stage 2 - mvn clean install 🡪 we use maven and generate .war file
3. Stage 3 - .war – jfrog 🡪 we push it to jfrog artefactory
4. Stage 4 - war -

Docker

1. Stage 1 - checkout pom.xml - as image
2. Stage 2 - we generate docker image
3. Stage 3 - push to docker hub/ecr
4. Stage 4 - container

Kubernetes

1. Stage 1 - checkout pom.xml - as image
2. Stage 2 - we generate docker image
3. Stage 3 - push to docker hub/ecr
4. Stage 4 - we deploy it to Kubernetes as pod

**Pipeline:**

* git checkout 🡪 build 🡪 push 🡪 deploy
* a Jenkins pipeline is a collection of jobs that automates the total software life cycle from version control to end users

**Jenkinsfile:**

Jenkins pipeline can be defined in a text file called as Jenkinsfile which is written in groovy DSL (domain specific/declarative language)

(artifactory (jfrog) integration with jenkins) 🡪 <https://youtu.be/BVxhLIfunmI>

sonarcube integration with Jenkins 🡪 <https://youtu.be/wn9wWYAShag>

https://github.com/bharathreddy06/SonarQube.git

**What are some of the default environmental variables in Jenkins?**

Some of the Jenkins environmental variables are:

* $JOB\_NAME - The name that you give your job when it is first set up.
* $NODE\_NAME - This is the name of the node on which the current build is running.
* $WORKSPACE - Refers to the path of the workspace
* $BUILD\_URL - Indicates the URL where the results of the builds can be found.
* $JENKINS\_URL - This is set to the URL of the Jenkins master that is responsible for running the build.
* as new Jenkins plug-ins are configured, more environment variables become available

**What is the use of the JENKINS\_HOME directory?**

* JENKINS\_HOME directory is the place where all the settings, logs, and configurations are stored. It stores all this information in XML files.
* Every directory has two subdirectories: builds and workspace., and some other files as well.
* the workspace directory is located at the place where Jenkins is building the project, and it contains the source code.
* The builds directory stores the history of all the builds performed for this job.

**CICD: 🡪 tomcat environment example**

pipeline {

agent {label 'slave1'}

stages {

stage('GIT Checkout') {

steps {

echo 'This is Git Checkout Stage'

git 'https://github.com/adhig93/java\_repo1'

}

}

stage('Build Stage') {

steps {

echo 'This is Build Stage'

sh 'mvn clean install'

}

}

stage('PUSH Stage') {

steps {

echo 'This is Push Stage'

}

}

stage('Deploy Stage') {

steps {

echo 'This is Deploy Stage'

sh 'sudo cp target/\*.war /opt/apache\*/webapps/'

}

}

}

}