**CI/CD**

source code --- build tools --- executable files language Build tool Executable

file C / C++ - make - \*.exe

Java - Maven/gradle/Ant/sbt/npm - jar, war, ear

Maven project -maven (pom.xml)

NodeJS project -npm (package.js)

Scala Project -sbt (build.sbt)

**build:** binary or executable file before testing

**release:** It’s a tested build ready to release to the customer

**sanity/bvt(build verification test):** it’s a basic functionality of a build that should never break

**hotfix/patch build:** It’s a critical fix which needs to be delivered to the customer within few hours. If I have modified 5 files among 1000 files , only 5 files will be compiled and regenerate the build. It takes less time.

**load build(Full build):** Here we compile source code form the scratch. It takes more time.

**maven lifecyle**

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

* **validate** - validate the project is correct and all necessary information is available
* **compile** - compile the source code of the project
* **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* **package** - take the compiled code and package it in its distributable format, such as a JAR.
* **verify** - run any checks on results of integration tests to ensure quality criteria are met
* **install** - install the package into the local repository, for use as a dependency in other projects locally
* **deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

**Day 30: CI/CD**

**what kind of issues you faced while building application (Build issue)?**

once the build is failed, we need to debug and identify why the build is failed.

If it’s a compilation issue, we need to work with the developers.

if it’s a build environment issue, we need to debug and fix it.

Environment may be

1. Memory is full on the server
2. CPU usage is high on the server
3. Dependency issue
4. version mismatch
5. Slave machine(Server) is not available in Jenkins job or if the server is down

JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

JAVA\_HOME=/usr/lib/jvm/java-17-openjdk-amd64

JAVA\_HOME=/usr/lib/jvm/java-21-openjdk-amd64

sudo update-alternatives --list java

to check different version avaialbfle

sudo update-alternatives --config java

**Day 30: CI/CD**

Note: we support deployment to QA, Dev, Stage and production environment

We deploy war file to the tomcat server using Jenkins job.

To deploy to QA, dev environment we need to take manager approval.

To deploy to the production environment, we need to take approval from SRE team/CAB(change advisory board) team

In our company we will do release once in a month

Jenkins:

* this is an open source CI and CD automation tool
* The default path of Jenkins is /var/lib/jenkins

<https://www.jenkins.io/doc/book/installing/linux/>

Refer to long-term support release commands to install the jenkins

Note: By default, Jenkins will run on 8080 and if you want to change the port number

On Master Jenkins server  
cd /etc/default/jenkins  
vi /usr/lib/systemd/system/jenkins.service  
sudo vi jenkins.service [HTTP\_PORT=6060]  
sudo systemctl daemon-reload  
sudo systemctl restart jenkins  
sudo systemctl status jenkins

**Continuous integration (CI)**

Integrate the changes from development team continuously without manual intervention. we can give quick feedback to the developers for there changes

**Continuous Deployment (CD)**

Tested build should be deployed to the production environment as early as possible which we call it as a continuous deployment

**Continuous Delivery (CD)**

Each change from dev team should be built and deployed to the test environment to test the changes

**Day 31: CI/CD**

Which are the different types of jobs you’ve worked on

1. Freestyle
2. Pipeline
3. Multibranch pipeline
4. Github organization

How to add credentials in Jenkins?

1. click on manage Jenkins, click on credentials
2. click on add credentials
3. select the kind, it might be username and password, ssh (username with private key), secret file, etc..,
4. Add the details and click on add credentials

Note: we require admin access to add the credentials

**Build triggers:**

1. Build Periodically

It will trigger the build based on the time, whether commit or no commit

2. Poll SCM

It will trigger Jenkins job based on the commit, if there are any commit within specific time, it will trigger Jenkins Job.

3. Webhooks

It will trigger the job based on the commit and push in the git repo.

4.Build after other projects are built  
It will trigger the job after the specific project has been built

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Day 32: CI/CD**

**how do you secure Jenkins?**

* we used matrix-based security. It will provide checkboxes, whichever the checkbox we check user will have those permissions

**how do you install plugins?**

* click on manage Jenkins, click on manage plugins, under the advanced settings tabs we need to upload a plugin file to install the plugins
* we need to download the Jenkins plugin file from Jenkins update centre(Jenkins.io)
* the extesnsion is hpi

A screenshot of a computer

AI-generated content may be incorrect.

**How to install the plugins?**

* We will download the plugin from Jenkin update server/centre (updates.jenkins.IO)
* We will add the plugin from manage Jenkins, manage plugins in advance tab we are going to upload a file, and install without restart.

How are you going to manage Jenkins master or what will you do if the Jenkin master goes down?

* We are using primary and secondary master and whenever the primary master goes down, then the secondary master will be running. Both the primary and secondary master controlled by Jenkins operation centre.

**Which are the different types of plugins which you have worked?**

1. parametrized plugin   
   -- this is used to pass the parameters to Jenkins job.  
   --I configure parametrized plugin for deployment job  
   -- deployment job asks parameters such as cluster name, artefact name an namespace name. These values will be passed to the deployment script through Jenkins job
2. gearman plugin  
   -- It’s a high availability plugin if the Jenkins master goes down, Jenkins will be upon running automatically on another server. Jenkins will be highly available.  
   -- gearman plugin will allow us to add other server details. When Jenkins is running on first master both the servers will be in sync. As soon as Jenkins masters goes down, Jenkins will be upon running on other server.
3. upstream plugin
4. downstream plugin
5. sonarQube
6. artifactory plugin
7. publishorcifs  
   -- this is userd to copy files from linux to windows

sonar qube

* It is used to check the quality of source code, if the quality code is more than 70% then we consider the quality of source code is good
* we integrated sonar qube with Jenkins, sonar qube works as a vulnerability tool in terms of scanning the source code.
* The unit test case results would be uploaded into sonar queue, upon the unit test case results, we will get matrix from quality profile and quality builds
* The default port is 9000

How do you take Jenkins backup?

* we need to take a backup of the jobs from the path /var/lib/jenkins and store in a separate repository in github

**Assignment:**

**Add the email plugin and configure it. Try to send the emails for build failure and build success.**

https://www.youtube.com/watch?v=j7jmXYZK5dU

**Day 33: Jenkins**

**Pipeline Job:**

Groovy script

<https://www.jenkins.io/doc/book/pipeline/syntax/>

<https://www.jenkins.io/doc/book/pipeline/syntax/#parameters>

Jenkins pipeline/ Jenkinsfile

* this is the place where we put all the job configurations to achieve multiple instance execution in a single pipeline job. By default, it will take one executor until unless we configure more executor.
* Jenkinsfile is written using Groovy script. We can define script in 2 different types  
  1. Declarative pipeline  
  2. Scripted Pipeline

Declarative pipeline

* Here we define the code under the pipeline tag
* syntax  
  pipeline { }

|  |
| --- |
| pipeline {  // code here } |

* to run the jobs on the specific server, we are defining through agents

Scripted Pipeline

* The code should be written under the node tag
* syntax  
  node { }

|  |
| --- |
| node {  // code here } |

* to run the jobs on the specific server, we are defining through nodes

Agent:

* it’s a block used to define the server where jobs should run
* we have 4 types of agents  
  1. Any  
  2. None  
  3.Label  
  4. Docker

any

* agent as any is always defined at the pipeline level
* we cannot define any at the stage level
* Jobs will pick the server to run whichever the server is available based on the load
* syntax  
  agent any

|  |
| --- |
| pipeline {  agent any  stages {  stage('Example') {  steps {  echo 'Hello World'  }  }  }  } |

none

* If we define agent as none at pipeline level, then it’s mandatory to define the agent on each and every stages
* syntax  
  agent none

|  |
| --- |
| pipeline {  agent none  stages {  stage('Example') {  agent any  steps {  echo 'Hello World'  }  }  }  } |

label

* if I want to run the jobs on specific server or on specific group then go with label
* syntax  
  agent { label ‘slave name/labelnamed }

|  |
| --- |
| pipeline {  agent { label ‘testlabel’ }  stages {  stage('Example') {  steps {  echo 'Hello World'  }  }  }  } |

or

|  |
| --- |
| pipeline {  agent none  stages {  stage('Example') {  agent { label ‘testlabel’ }  steps {  echo 'Hello World'  }  }  }  } |

docker

* Here we are pulling the docker image to run the jobs
* syntax  
  agent { Docker : imagename }

**How do you run the different stages on different servers?**

* By using agent none at the pipeline level and then use agent label at each stage.

|  |
| --- |
| pipeline {  agent none  stages {  stage('Example') {  agent { label ‘testlabel1’ }  steps {  echo 'Hello World'  }  }  stage('Example1') {  agent { label ‘testlabel2’ }  steps {  echo ‘Welcome to the devops classs’  }  }  }  } |

Practice:

|  |
| --- |
| pipeline {  agent any  stages {  stage('Test1') {  steps {  echo "Hi"  }  }  stage('Test2') {  steps {  echo "Welcome"  }  }  }  } |

|  |
| --- |
| pipeline {  agent any  stages {  stage('checkout') {  steps {  sh 'rm -rf hello-world-war'  sh 'git clone https://github.com/phaninandigam/hello-world-war.git'  }  }  stage('build') {  steps {  sh 'cd hello-world-war'  sh 'mvn clean package'  }  }  }  } |

|  |
| --- |
| pipeline {  parameters {  string(name: 'cmd', defaultValue: 'package', description: 'Who should I say hello to?')  choice(name: 'ch', choices: ['One', 'Two', 'Three'], description: 'Pick something')  }  agent any  stages {  stage('checkout') {  steps {  sh 'rm -rf hello-world-war'  sh 'git clone https://github.com/phaninandigam/hello-world-war.git'  }  }  stage('build') {  steps {  sh 'cd hello-world-war'  sh 'mvn clean $cmd'  }  }  }  } |

**Day 33: Jenkins**

**Difference between Pipeline job and multipipeline job**

**Pipeline**

* In normal pipeline job is meant for building a specific branch from the SCM and it deploy respective environment. The pipeline job supports both pipeline steps to be added in Jenkins configuration and form the CSM. We can use the pipeline job for adhoc jobs parameterised job execution and to debugging pipeline as a code.  
    
  **Multibranch pipeline**
* it’s meant for building multiple branches from a repository and deploy respective environments like QA, UAT, Prod, Dev

NOTE: Don’t use multibranch pipeline if you don’t have a standard branching and CI/CD strategies

**assignment**

create a pipeline for hello world war project. It should consist of checkout, build and deployment

post build send an email

<https://github.com/phaninandigam/hello-world-war/blob/master/Jenkinsfile>

A screenshot of a computer

AI-generated content may be incorrect.

maven repo:

* while budling maven search dependencies in local repo(.m2), if it doesn’t find the dependencies then it will search in central repo (Jfrog, Artifactory)
* If it doesn’t find in the central repo, then it will search in remote repo (Default URL of the dependences)

Do you write pom.xml?

* pom.xml is given by the dev team, we mention path for local dependences
* Normally we edit settings.xml under .m2 folder to change path for dependencies

why are slave machines required?

* master slave are used to distribute load to different servers and also we can run specific job on the specific servers
* example: we can compile java source code on the server which provides java environment

NOTE: Normally we will never run the jobs on master, we will run on slaves

* Normally 10-15 slaves will be there
* we will uses labels to run on available slaves

**Day 34: Jenkins**

**Day 35: Jenkins**

**Shared Libraray**

A Shared Library in Jenkins is a reusable collection of Groovy scripts and functions that can be shared across multiple pipelines.

It allows you to centralize and standardize common pipeline tasks, such as building, testing, deploying, or setting up environments.

Key Features of Shared Libraries:

Reusability: Write once and use it in multiple pipelines.

Modularity: Organize reusable into functions or classes for better maintenance.

Consistency: Ensure consistent behavior across pipelines by using standardized scripts.

Scalability: Simplify pipeline management for large-scale projects with many jobs.

Shared Library Function: Install Java and Maven

vars/installJavaAndMaven.groovy

|  |
| --- |
| def call(String javaVersion = '17', String mavenVersion = 'latest') {  echo "Installing Java ${javaVersion} and Maven ${mavenVersion}..."  // Update package index  sh 'sudo apt update'  // Install the specified version of Java  sh "sudo apt install -y openjdk-${javaVersion}-jdk"  // Install Maven (either a specific version or the latest)  if (mavenVersion == 'latest') {  sh 'sudo apt install -y maven'  } else {  echo "Installing Maven version ${mavenVersion} is not implemented in this example."  error "Currently only 'latest' Maven installation is supported."  }  // Verify installations  sh 'java -version'  sh 'mvn -version'  echo "Java ${javaVersion} and Maven installation completed!"  } |

Shared library git link

<https://github.com/phaninandigam/jenkins-shared-libraries-.git>

A screenshot of a computer

AI-generated content may be incorrect.

If we look at the below image, we are seeing a groovy script in the name of checkoutcode.groovy and inside the file we have a call().

Whenever the file is called, the call() will be loaded.

A screenshot of a computer

AI-generated content may be incorrect.

We use groovy script to write the pipelines scripts.

Below is an one of the custom groovy which I have written.

Note: While creating a new groovy script, please make sure not to use the keywords like pipeline.groovy.  
Same works for methods as well, don’t create methods with checkout(), run(), build().

A screenshot of a computer

AI-generated content may be incorrect.

Using shared library in other repos. we are using @main which represent the groovy is in main branch.

A screenshot of a computer

AI-generated content may be incorrect.

**Day 36: Jenkins**

**Artifactory Tool:**

**JForg**

it’s a tool which is used to store the artifacts (executable/jar files)

on ec2 instance, we need t2.medium to install the JFrog

**Note: Please take a look at the JFROG setup guide.doc file for more information**

**Day 37: Sonar qube**

Setup sonar qube

Day to Day activities on Jenkins

* Writing the pipelines based on the requirement
* Taking backups
* Montoring the jobs using tools like Grafana