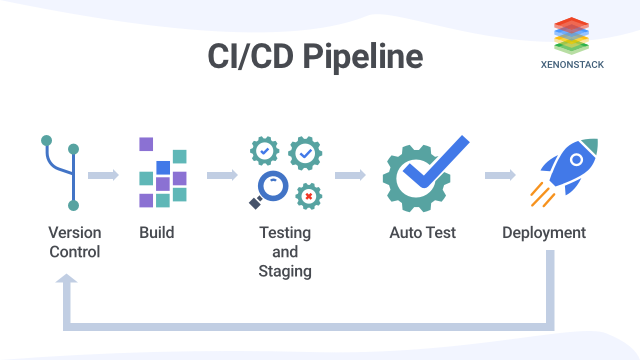
Jenkins

**01. Jenkins Section Introduction**

**Continuous Integration (CI):** CI 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.

If the build is not a green system notify the developer immediately. By this developers can detect errors quickly and locate them more easily.



Benefits

Backtrack issue easily, less time to debug, deployment can be done at any time,

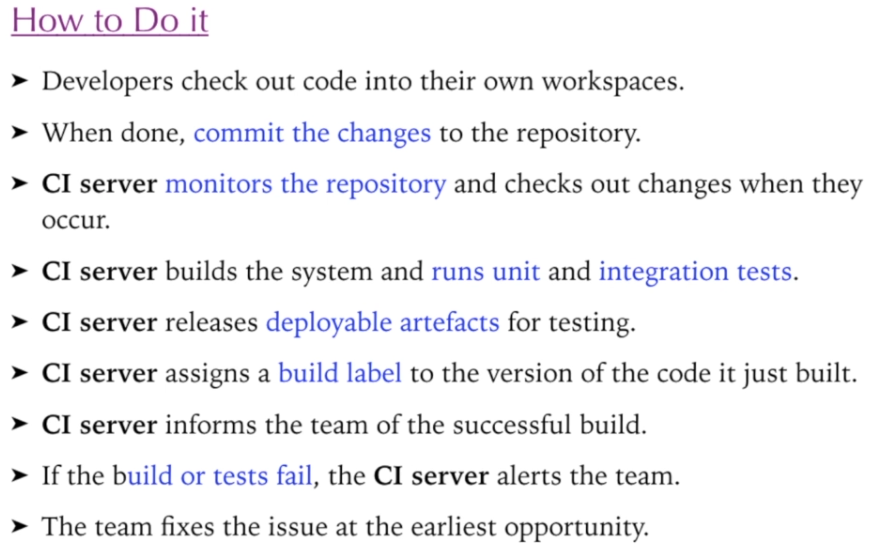
Practics

Maintain a single source repository

Automate the build

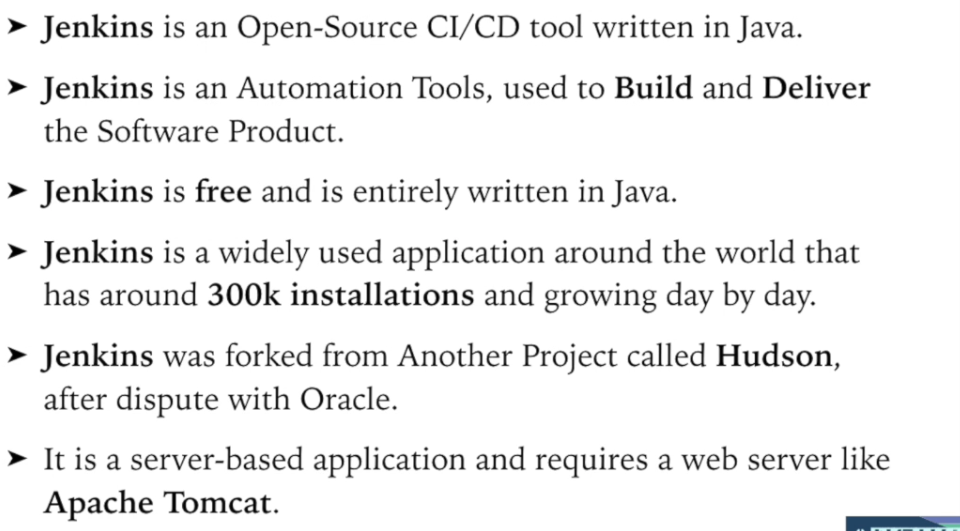
Make your build self-testing

Automated deployment



**Continuous development** (CD) of code to an environment once the code is ready to ship

By adopting both CI and CD you not only reduce risks and catch bugs quickly but also move rapidly to working software.



<https://www.jenkins.io/download/> to download Jenkins. Select generic java package

goto your Jenkins war directory. Then open the command prompt and run bellow command to start Jenkins

java -jar jenkins.war --httpPort=9090

set details

user id : bibhu082

pwd: study\_pwd\_all

[bsp@gmail.com](mailto:bsp@gmail.com)

**02. Jenkins Getting Started | Get Jenkins Ready**

Master

Schule the job

Dispatch build to the s slave for actual job execution

Monitoring the salve and recording the build results.

Slave/Node

Execute builds job dispatch by master.

Slave are the computers/vm that are setup to build projects for a master.

Jenkins run separate programme called slave agents on slave.

When slaves are registered to a master, a master start distributing the loads to slave

Jenkins Job

Job is referred to runnable tasks that are controlled and monitors by Jenkins.

Jenkins Executor

Separate stream of builds to be run on a node parallel.

A node/slave can have one or more executor

Jenkins Plugin

Is a piece of code that extend the core functionality of the core Jenkins server

Create Jenkins JOB

Go to localhost:9090, login to Jenkins

Click on New Item then enter item name “First Jenkin Job” then select freestyle project. Then click on ok. Then add some custom description.

Select Throttle builds 1

Select Delete workspace before build

Select Add timestamps to the Console Output

Select build the execute shell

#Start of for loop

for a in 1 2 3 4 5 6 7 8 9 10

do

#if a is equal to 5 Sleep the loop

if [ $a == 5 ]

then

sleep 10

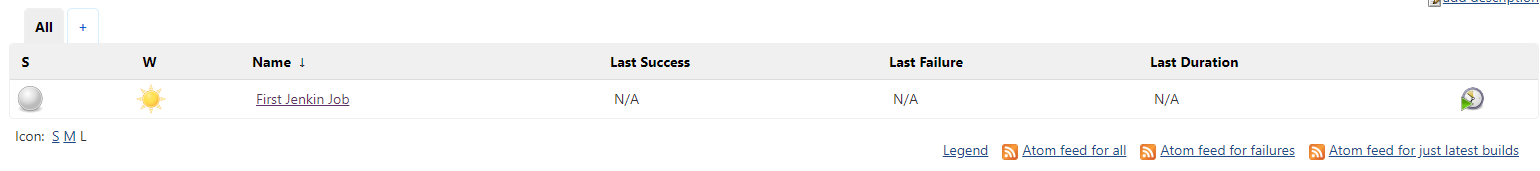
fi

#print the value

echo "Iteration number $a"

done

Now save and go to home Jenkins



Set sh.exe for windows. Jenkins Home -> Manage Jenkins -> Configure System -> Shell -> shell executable -> C:\Program Files\Git\bin\sh.exe

**03. Continuous Integration with Jenkins**

Integration with Github

Go to Jenkins home then manage Jenkins then manage plugin then install github integration. Select install without restart.

Configure Jenkins with Git and Maven

Go to Jenkins home then manage Jenkins then global tool configuration

Add jdk name java1.8 location C:\dev\_env\softwares\java-8u251

Then add git name git location C:\Program Files\Git\bin\git.exe

Then add maven name maven location C:\dev\_env\softwares\apache-maven-3.6.3

Create our First Maven-based Jenkins Job

Go to Jenkins home add item then give project name First maven git project then select free style then give description. Give Throttle build as 5. Then source code management and select git. Give the location https://github.com/bibhusprasad/011\_jenkins\_code\_master.git . Then build environment delete workspace before build start. Then go to build invoke top-level maven targets select install maven then select goals test install. Go to advanced POM give pom.xml location i.e. maven-samples/single-module/pom.xml then save, and build the project.

Now edit the configuration set maven goal as clean package to generate the package and save build. Now the jar file will created inside C:\Users\Bibhu\.jenkins\workspace\First maven git project\maven-samples\single-module\target\single-module-project.jar.

Automate continuous integration

Source code polling in Jenkins

Go to project configuration then build trigger and select Poll SCM and schedule the job by providing corn. \* \* \* \* \*

\*(minute) \*(hour) \*(day of month 1-31) \*(month 1-12) \*(day of week 0-6 Sunday=0)

0 0 \* \* \* every day at mid night

0 2-4 \* \* \* every day 2am, 3am, 4am

0 2,4 \* \* \* every day 2am, 4am

Now new manu created [Git Polling Log](http://localhost:9090/job/First%20maven%20git%20project/scmPollLog) to track poll log details. If there is a change in master code base then it trigger a new build.

Remote Build Trigger or Other build trigger

Go to configure then build trigger select trigger build remotely give authentication token name MY\_AUTH

Copy Use the following URL to trigger build remotely: JENKINS\_URL/job/First%20maven%20git%20project/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME. now configure the trigger url i.e.

<http://localhost:9090/job/First%20maven%20git%20project/build?token=MY_AUTH>

Now a new build will be triggered.

Like this we can trigger periodically by providing corn job

**04. Continuous Delivery with Jenkins**

Archive in Jenkins is how to save file outside workspace

User can clean workspace, run other build and the file archived is safe

If you want get old file then you need to archive it

Go to Jenkins home then project configure then post build action then select archive the artefacts and give \*\*/\*.jar save and build

\*\* all available path inside your workspace

\*.jar any file name with .jar extension

Now click on job number now a new location created Build Artifacts

Install and configure tomcat

Download and unzip tomcat inside a location for me it is in C:\dev\_env\softwares\apache-tomcat-9.0.36

Then go to conf folder and open tomcat-users.xml and add/change below line

<role rolename="manager-script"/>

<role rolename="admin-gui"/>

<user username="tomcat" password="tomcat" roles="manager-script,admin-gui"/>

Then start tomcat go to bin folder open command prompt and hit startup.bat to start the tomcat and to shutdown use shutdown.bat

Deploy application Staging env

**Create a job to produce Tomcat Deployable Artifacts**

Go to Jenkins home Create a new view (optional). Click + sign near to All. Give a name like tomcat view and hit ok ok two time. Now new view created. Go to that tomcat view and click on new item give a name like tomcat package application select freestyle project and hit ok.

Then give any description select discards old builds select 5 and 5

Then go to git and give git url <https://github.com/bibhusprasad/012_jenkins_code_master>

Then select master branch

Then select delete workspace before build and add time stamp

Then go to build and select invoke top level maven targets select maven then goal clean package and give pom location java-tomcat-sample/pom.xml

Then go to post build action select archive to artifacts and give \*\*/\*.war

Now build your application.

**Install Copy artifacts & Deploy to Containers plugin**

Go to Jenkins then manage Jenkins then manage plugin and install above two plugin without restarting jenkins

**Deploy application to staging environment**

Crate a new job for deployment

Go to tomcat application view click new item and give a name like deploy application staging env and select freestyle project. Give a description. Then discard old build give 5 and 5. Then select delete workspace and add time stamp

Then go to build select copy artifacts from another project give project name tomcat package application and Latest successfully build check stable build only. Then Artifacts to copy give \*\*/\*.war

Then post build section select deploy war/ear to a container. Then War/ear file give \*\*/\*.war then context path give / then container select the tomcat version for me it is 9.x

Then add credential select Jenkins credential and give user name and password both as tomcat tomcat. We configured above. Now choose tomcat/\*\*\*\*\*

Then give tomcat url <http://localhost:8080/> and save and trigger build

Now go to tomcat home the application is deployed to tomcat

**Now configure for auto trigger the deployment process**

Select poll SCM corn job **\* \* \* \* \***

Go to tomcat package application then select configure then go to post build configuration select build other project and give [deploy application staging env](http://localhost:9090/view/tomcat%20application/job/deploy%20application%20staging%20env/) and select trigger only if is the build is stable

Now do some changes in index.jsp file and pusgh the changes to master. So poll SCm found the changes in git then it trigger a build then after build success post build trigger deployment build and deployment happen in tomcat server. Now refresh tomcat the changes will reflect in browser

Build Pipeline Plugin

Build pipeline plug provide a pipe line view of upstream and downstream connected jobs

The plugin gives the ability to form a chain of jobs based on the upstream/downstream dependencies

Go to manage plugin and install build pipeline without restart

Create a new view by click on + near to All then give a name like pipeline view application and select build pipeline view and select ok

Then give build pipeline view by providing any name like Deploy Application Staging Env

Then select initial job by selecting tomcat package application

Deploy to Production

Create a pipeline to deploy application in production

Suppose localhost:8080 is our dev environment and localhost:8081 is prod environment. First we need to setup prod environment. Then we need to configure tomcat first.

First copy paste tomcat installation directory and give a new name apache-tomcat-9.0.36-prod. Change <Server port="8006" shutdown="SHUTDOWN"> then Change the prod <Connector port="8081" protocol="HTTP/1.1" then change redirectPort="8444" />

Now start both tomcat server

Now create a new job for production deployment. Create a new item under tomcat application view and give name deploy application production env and select freestyle project and hit ok. Give any description and select delete workspace before build start and add time stamp.

Then go to build select copy artifacts from another project then give project name i.e. tomcat package application then select Latest successful build and select checkbox Stable build only then select Artifacts to copy \*\*/\*.war and save.

Then go to post build action and select deploy war/ear to a container. Then select WAR/EAR file as \*\*/\*.war context path as / container select tomcat/\*\*\*\*\* credential and tomcat url as <http://localhost:8081/>

**Now we have to configure if staging build is success then we need to deploy production build**

So for that go to deployment application staging env then select configure and post build action select build other project manual step. And give the downstream job name i.e. [deploy application production env](http://localhost:9090/view/tomcat%20application/job/deploy%20application%20production%20env/)

Go to pipe line view to check the GUI view. Now do some changes in index.html file and push the changes. So now project pipeline will pick the changes from master and do a build. Then downstream staging env build will trigger and deploy application in dev server. But not in prod server as manual deployment is selected 8080 has update changes but 8081 has old changes. To deploy in prod we need to manual select build in prod job

**05. Infrastructure as a code**

Infrastructure as code is the ability to programmatically manage and provision an entire technology stack through code, rather than using manual and complex process.

Version control provide the History and Audit trail.

Easy rollback to prior versions job and builds

Allow developer to bundle build instruction with their application code

**06. Jenkins Job DSL**

Jenkins DSL (Domain specific language) is programmatical way to implement Jenkins job.

Jenkins job DSL plugin is used to define jobs in programmatical form.

Jenkins job can be described as Groovy based script

User can create multiple jobs with Jenkins DSL plugin and it comes with lots of benefit like version control, audit history, peer review, roll back and backup

Manage plugin -> install job DSL plugin

Seed job

To use job DSL plugin we need to create a seed job. The seed job is Jenkins job which runs a DSL script and then generate a new job. The seed job is a normal free style job that you add the “Process job DSL” build step. The step takes the DSL and generate the configured job.

<https://jenkinsci.github.io/job-dsl-plugin/>

Create a groovy file name MavenProjectDSL.groovy and add it to master

job('First-Maven-Project-Via-DSL'){  
 description("First Maven job generated by DSL, the project is a small Maven project hosted on github")  
 scm {  
 git("https://github.com/bibhusprasad/012\_jenkins\_code\_master.git", master)  
 }  
 triggers {  
 scm('\* \* \* \* \*')  
 }  
 steps {  
 maven('clean package', 'maven-samples/single-module/pom.xml')  
 }  
 publishers {  
 archiveArtifacts '\*\*/\*.jar'  
 }  
}

Create new item maven seed job select free style project then give description then discards old build select 5 and 5. Select git give git http location and master branch. Then go to build give DSl groovy file name MavenProjectDSL.groovy and save the project.

Then build the job it will fail first time due to groovy script not yet approved due to security reason. For that we need to go manage Jenkins then in process script approval and approve the script. then go to job and do build again.

Every time there is a change in groovy script file, we need to approve that.

**07. Jenkins as a Code Pipeline (DSL)**

Jenkins code pipe line uses DSL

DSL script via groovy and so it is extremely flexible

DSL use automate the Jenkins job

Having the build and deployment **Pipeline as Code** **(PaC)** build on the benefit of having **Everything as Code (Eac)** e.g. immutable version control, audit trails, peer review, textual representation, and knowledge sharing

At a high level, a pipe line consist of three parts i.e. **Build and testing the artifact, assuring quality and orchestrating deployment in production**

Build and testing the artifact

This include compile the code, runs initial tests and output an artifact for storage in a repository

Assuring quality

This part of pipeline may run additional process across the code database – think security, cyclomatic complexity, bug detection, code coverage metrics

Orchestrating deployment in production

Means deploy application on dev, staging, production. And quality assurance stage also runs against each of the environment as the deployment is orchestrated.

Jenkinsfile

Jenkinsfile is a text file that stores the entire workflow as code and it can be checkout into a SCM on your local system

The Jenkinsfile is written using groovy DSL

|  |
| --- |
| pipeline { |
|  | agent any |
|  | stages { |
|  | stage('Init') { |
|  | steps { |
|  | echo 'Hi, this is Anshul from LevelUp360' |
|  | echo 'We are Starting the Testing' |
|  | } |
|  | } |
|  | stage('Build') { |
|  | steps { |
|  | echo 'Building Sample Maven Project' |
|  | } |
|  | } |
|  | stage('Deploy') { |
|  | steps { |
|  | echo "Deploying in Staging Area" |
|  | } |
|  | } |
|  | stage('Deploy Production') { |
|  | steps { |
|  | echo "Deploying in Production Area" |
|  | } |
|  | } |
|  | } |
|  | } |

Pipeline is a user defined block which contains all process such as build, text, deploy. It is a collection of all the stages in a Jenkinsfile

Node is a machine that executes and entire workflow

Agent is a directive that can run multiple builds with only one instance of Jenkins. It instruct Jenkins to allocate an executor to build.

Any Runs the pipeline or stage on any available agent

Stages This block contains all the work that need to be carried out. There can be more than one stage within this directive. Each step performs a specific task

Steps A series of steps can be defined with in stage block. These steps are carried out in sequence to execute a stage. There must be at least one step within a step’s directive.

Create a new item name Sample code pipeline and select pipeline and ok. Give description and discard old build. Then go to pipeline and define pipeline script from SCM. Give git url and give Jenkins file name and save and do a build

