**Jenkins**

Trainers Docs: <https://docs.google.com/document/d/1RuUq8jPB0Vv9HZAdV4Cm69KrHXAcV6tLePTkJavf1pQ/edit>

Interview: <https://www.simplilearn.com/tutorials/jenkins-tutorial/jenkins-interview-questions>

**What is CI/CD**

**What is Jenkins**

**Installtion of Jenkins**

**Architecure of Jenkins**

**Node in Jenkins**

**Linked with DevOps Tool**

**What is PipeLine**

**Integration with Pipeline**

**What is Continuous Integration(CI):**

The process of having shorter release cycles i.e creating small features and integrating them to the source code, and employing automated build and test process for quicker feedback is called continuous Integration.

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.

Advantage of of CI:

1. Frequent Commit hence small feature release
2. Automated Build & testing
3. Instant feedback to developer
4. Low Risk & Daster Delivery

CI/CD Piple line tool: Jenkins, TeamCity, Bamboo, Github Actions, Gitlab, aws code pipeline

**What is Jenkigs:**

Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies. Jenkins is an open source automation server tool to build & test software project continously. With the help jenkins you can accelerate software development.

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.



**Let's see how Jenkins works**. The above diagram is representing the following functions:

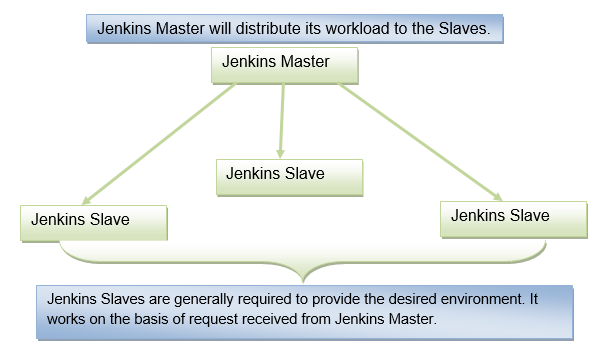
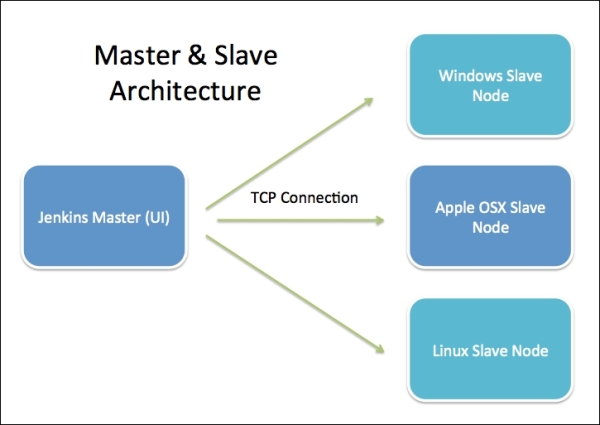
* First of all, a developer commits the code to the source code repository(git). Meanwhile, the Jenkins checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
* If the build fails, then the concerned team will be notified.
* If built is successful, then Jenkins server deploys the built in the test server.
* After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
* It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

**Jenkins Architecture**

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

* Jenkins Master/Server
* Jenkins Slave/Node/Build Server



### Jenkins Master

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

The server's job or master's job is to handle:

* Scheduling build jobs.
* Dispatching builds to the nodes/slaves for the actual execution.
* Monitor the nodes/slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master/Server instance of Jenkins can also execute build jobs directly.

### Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.

**Installtion**:

Debian Stable : https://pkg.jenkins.io/debian-stable/

1. **Create Server instance on AWS and connect to server which you created through putty/mobstrem/aws terminal**
2. **Update & load the system image**

sudo apt update -y

1. **Jenkins is written in java, so 1st we need to install java(Jenkins written in java)**

sudo apt-get install openjdk-11-jre -y

1. **Add GPG key**

curl -fsSL<https://pkg.jenkins.io/debian-stable/jenkins.io.key> | sudo tee \

    /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

    https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

    /etc/apt/sources.list.d/jenkins.list > /dev/null

1. **Again update the package & load image**

sudo apt update -y

1. **Install the jenkins**

sudo apt install jenkins -y

1. **Start the Jenkins server**

sudo systemctl start jenkins

1. **Enable the jenkin service**

sudo systemctl enable jenkins

1. **Check the status of jenkin service**

**sudo systemctl status jenkins**

1. **Open the browser and type your public ip flowed by jenkin port(:8080)**=>Jenkins Default port: **8080**

**13.113.254.163:8080**

1. **Get Password by typing below command (**Jenkins user password store in Jenkins/secrets/ directory **)**

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

1. **Optional Extra command & it’s not required => check which port is used by which service**

**sudo netstat -ntlup**

1. **Create User with details if install & open jenkins 1st and login**

**Task/Practice/Exercise with Jenkins**

1. Pull the code from github
2. Docker build to create docker image(install & build the docker image)
3. Push image to dockerhub (docker image repository)
4. Deployed to k8 cluster ()

**How to Create & Setup Job/Task/Project**

**Example-1 => Pull the code & Build it**

1. Click on **+ or new Item or create a job**
2. Name of job **MyFirstJob**
3. Select your job option like **freeStyle Project**
4. click on **OK**
5. Now you on Configure page for configure your project/job/task
   1. **General Section**: where you can put about your project descriptions
   2. **Source Code Management**: Where & which branch you want to get/pull/push the code
   3. **Build Triggers** : How your build will trigger and what all those command will run to build your project
   4. **Build Environment** : Environment for your project
   5. **Build Steps** :
   6. **Post Build Actions** : What action do you want after/post build
6. Go To **Source Code Management :**
   1. Select **Git** and there is four sections
      1. Repository
      2. Branch to Build
      3. Repository Browser
      4. Additional Behaviors
   2. Put your git repo URL**:**  <https://github.com/abhishekraics001/myfirstrepo-public.git>
   3. Add git credentials if your project is hosted as private repo & not required for public branch
   4. Add your branch(**\*/master**) in which you want to build the project
   5. Build The Task

**Example/Task- 2**

1. Pull the code from git repo
2. Build the docker imgae
3. Push it to docker hub
4. Deployed it to k8 cluster

**Step to Do above task**

**Step-1 : Basic setup of Project**

1. Create **New job** or New Items
2. Select the Project type: **freestyle project**
3. Select **git** add add below repo with master branch

<https://github.com/abhishekraics001/demo-app.git>

or

<https://github.com/abhishekraics001/HTMLStaticApp>

1. **Save & build**

**Step- 2 : Install the docker**

1. Then we need to **install** some **tool(docker, k8s, plugging on same )**
   1. sudo apt update –y
   2. sudo apt install docker.io –y
   3. sudo systemctl start docker
   4. sudo systemctl enable docker
   5. sudo systemctl status docker
2. At time of install the jenkin we add one user like admin/jenkins and all task/command are running by jenkin user and to view all user which are ragister with our server

cat /etc/passwd

Or

cat /etc/passwd | grep jenkins

1. How to check which user are currently login in our system server

whoami

1. To run any command from Jenkins tool =>
   1. Go To Configure
   2. Select Build scrip
   3. Select the **Execute shell**
      1. Type your command whatever you want to run & execute

**Whoami**

**Docker ps**

1. So we need to **Give privilege permission to jenkins user** to run any shall/linux command in docker
2. Note: Jenkins job are execute by **jenkins or super user** only
   1. Option 1: you add sudo command before executing any command
   2. Option 2: you can add yourusername(jenkins) in docker group through below command
      1. sudo usermod jenkins –G docker
      2. sudo systemctl restart jenkins
      3. And logout from jenkin dahsborad & login again
3. **Jenkins Home Directory** :

/var/lib/jenkins

1. In Jenkins **every job have workspace** where code is copy/cloned by jenkins:

/var/lib/jenkins/workspace/YourJobName

**Step-3: To create and build the docker image through Jenkins script shall**

1. In order to create & build the image 1st we need to add **Dockerfile** file in your/our project root directory list of instruction/command /steps
   1. Like below: (yourProjectFolder/Dockerfile)

**FROM nginx**

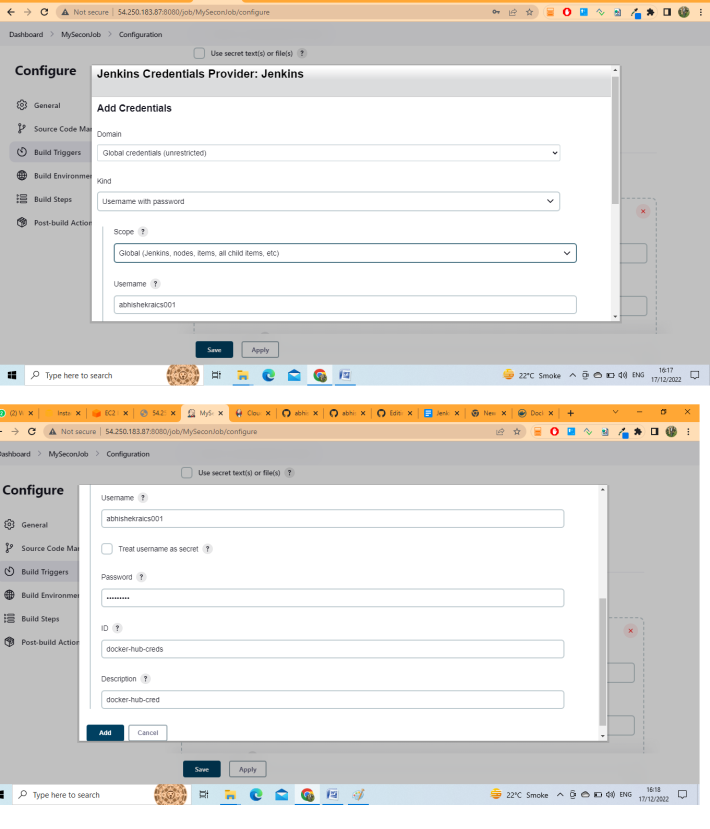
**COPY . /usr/share/nginx/html**

**EXPOSE 80**

* 1. **Docker Help command(Below command for manual purpose only)** which execute inside jenkins workspace job directory (/var/lib/jenkins/workspace/YourJobName)
     1. docker build –t myimage
     2. docker login
     3. docker tage
     4. docker push
  2. Do it by Jenkins Build step that run automatically at time of building the code
     1. Go To **Build step & select Execute Shell** (where you can write any linux command to done/perform your task)
     2. **Add** below **command for docker to create & build docker image from dockerfile**
        1. docker build –t myimage .
        2. docker image ls

**Steps-4 : Plugging Installation:**

1. **Plugging installation in Jenkins for create, build & push docker image on dockerHub.**
   1. To perform any type of administration task then we have options manage jenkins
   2. Go To **Dashboard -> Manage Kenkins -> Plug-in Manager**
      1. **Available plugging**: you can find all plugging which available for jenkins
      2. **install plugging**:
   3. Installing the docker plugging: **CloudBees Docker Build and Publish** (<https://plugins.jenkins.io/docker-build-publish/>)
   4. **Remove** the **build step command** if any for docker
   5. **Click on Build Step options** and Select the options => **Docker Build and Publish**
   6. After click/Select the options we need to **fill some information** about docker repository
      1. 1st Go to DockerHub and create an account if you have not setup yet => <https://hub.docker.com/>
      2. **Registry** **Name**(yourUserName/dockerImageName): **abhishekraics001/myfirstdockerimage01**
      3. Add your **Image Tag**(like V1): **V1**
      4. Add Your **Registry(docker) credentials**: by clicking on Add or select if already save in jenkins
         1. Kind: user name with password
         2. Scope: Global
         3. UserName
         4. Passord
         5. Id & Discriptions



f. **Save & Build**

**How to Integrate K8s with/in Jenkins**

1. **1st install & configure the K8s cluster(master nodes & workers nodes)**
2. Type below command to check the status of docker & K8s
   1. **kubectl get nodes**
   2. **kubectl get service**
   3. **kubectl get pods**
   4. **kubectl get deploy**
3. **kubeConfig file**: Kubernetes uses a YAML file called **kubeconfig** to **store cluster authentication information for kubectl** . kubeconfig contains a list of contexts to which kubectl refers when running commands. By default, the file is saved at **$HOME/. kube/config** . A context is a group of access parameters.
   1. Get the join token from **config** file through below command on( **master** **node** to **get the join token)**
      1. **cd .kube/**
      2. **cat config**
      3. Copy the all connect/token which are displaying after run the cat config command and pest it on worker/jenkins worker/slave node
4. **After that we have need install kubectl(k8s) on jenkins server**
   1. Note: make sure to instastall every plugging or software on jenkins server as well to perform any software/dependency/plugging base task(like: docker, k8s, etc)
   2. To install docker: we have already done in above section
   3. To **install k8s on Jenkins server** run the **below command**

Step- 1 : Add the **gpg** by copy & pest in single command

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg <https://packages.cloud.google.com/apt/doc/apt-key.gpg>

Step-2: Adding Repository of k8s by copy & pest in single command

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

Step- 3: **update the apt**

sudo apt update –y

sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y

Step-4**: Install the Kubectl**

sudo apt install kubectl -y

Step -5: Check your kubectl install version

**Kubectl version**

Step-6: Setup the kubeconfig file on jenkins/worker/slave node(which store inside: at **$HOME/. kube/config**) & generate the join token to communicate with master/cluster node. 1st get kubeconfig configuration from master node and add it in worker/slave node.

Note: jenkins run & execute command with jenkins user so that we give permission/privilege/admin permission to jenkins user to perform & execute the command.(So we need to add jenkins user in docker group)

* + - * 1. **Add Use user as super user:**

sudo su => 1st login with super user

su – jenkins => to login as Jenkins super user

whoami=> to check currentally which user login

pwd => to check the current path of jenkins user

kubectl get pods => you will get parmission error bocz jenkins user don’t have parmission to run & execute this command

kubectl get nodes => you will get parmission error bocz jenkins user don’t have parmission to run & execute this command

* + - * 1. **Add kubeconfig key In** Jenkins user directory or through Jenkins user after add make **jenkins as super user:**

mkdir .kube => crating .kube directory

cd .kube/ => Moving/going to to .kube directory

vim config => to open the config file through vim editor

Pest the all content which you have copy from master node

Note: below command to get join token from master node or master machine

sudo su => login as super user

cd => go to home directory

cd .kube => got to .kube directory

cat config => view the private & public key

VIM/Cat text Editor: Press or Enter the keyword in Editor

* + - * 1. I=> to enable edit mode
        2. Esc => to exit from edit mode
        3. **:wq** => for write & quit the editor
        4. **:q** => quit only
        5. **:rq** => read & quit

Run below command to check your machine cluster status

kubectl get nodes => To get list of connected node

kubeclts get services => To get list of connected services

kubectl pods => to get list running pods

Add Your Kubernets plugin in Jenkins

Search & install **Kubernetes CLI**  in available plugins section

Install & restart

Go to **Your Job => Configuration => Build Environment** and you can see there is option of **Configure Kubernetes**

Note: for right now we are not going to configure k8s through Jenkins. To check the configuration, how many k8s node, service, pods are running on Jenkins server you can type below command in your job

**Configure** **=> Build** **Steps => Execute Shall** command and save & **Build Now**

1. kubectl get nodes
2. kubectl get pods
3. kubectl get services
   * + - 1. Create two file one for k8s deployment **(development.yml)** and one for **service.yml** file

Create one **folder** **k8 in your project root directory and create two file deployment.yaml and service.yaml**

**deployment.yaml**

1. apiVersion: apps/v1
2. kind: Deployment
3. metadata:
4. name: mydeployment
5. spec:
6. strategy:
7. type: RollingUpdate
8. replicas: 2
9. selector:
10. matchLabels:
11. app: frontend
12. template:
13. metadata:
14. name: myPod
15. labels:
16. app: frontend
17. spec:
18. containers:
19. - name: myweb
20. imagePullPolicy: Always
21. image: abhishekraics001/myfirstdockerimage01
22. ports:
23. - containerPort: 80
    * + 1. **service.yaml**
24. apiVersion: v1
25. kind: Service
26. metadata:
27. name: myfrontend-service
28. spec:
29. type: NodePort
30. ports:
31. - port: 80
32. targetPort: 80
33. nodePort: 30000
34. selector:
35. app: frontend

After that open the **configure => Build Step => Execute shell** and type the below command

* + 1. kubectl create - f k8/deployment.yml => K8 is folder name where have put the deployment.yml file (Optional-> if any error coming remove this one)
    2. kubectl create - f k8/service.yml => K8 is folder name where have put the service.yml file (Optional-> if any error coming remove this one)
    3. kubectl get all => For all command, create, build & push the image

How to Add dynamic Variable in Jenkins:

* + - 1. Go to Configure => General =>
      2. Select This Project is parameterized
         1. Select String Parameter

Name: add /Enter your parameters Name

image\_version

Default Value: Add your Default value & Save it

V1

Put your parameters name followed by $ sysmbol in your build section for tag or any like below

$image\_version

After that you can see new Option in your Jenkins => Build the code by clicking

Build with Parameters

* + - * 1. Select the choices Parameters

Add Name:  **Branch**

Add your choice in box like drop down

* + - 1. Developer
      2. Testing
      3. Staging
      4. Production
      5. Release

* + - 1. Help Command:

a. kubectl get deployment => Get the list of k8s deploy

b. kubectl get pod => To get the list of pod running

c. docker images => get the docker image list

d. kubectl get nodes -o wide => get the describe of k8s nodes

e. kubectl describe pod

g. docker rmi -f nginx => For delete the docker image

h. docker run -d nginx => Pull the nginx image from docker hub

i. docker image ls => Display the list of image

j. kubectl get deploy => Get the list of k8s deploy

k. kubectl delete deploy YourDeploymentName => To delete the k8s deployment of

**Class:3**

**If else condition with parameters variable in Execute shell command**

Supposed I have create one Boolean type of **parameterize variable** in with name of **isTesting**

Write the command in **Execute shell** like below

if $isTesting == true

Do this

else

do this

**How to check deployment is running or not in shell Execute sections**

Shell scripting –

<https://www.digitalocean.com/community/tutorials/if-else-in-shell-scripts>

status=$(kubectl get deploy | grep mydeployment | wc -l)

if [ $status -eq 1 ]

then

  echo "Deployment exists in the cluster"

  kubectl set image deployment/mydeployment myweb=abhishekraics001/appimage:$image\_version

else

  echo "Deployment Doesn't exists"

  kubectl create -f k8/deployment.yaml

fi

status = $ (kubectl get deploy | grep mydeployment | wc -l)

if [ $status -eq 1 ]

then

  echo "Deployment exits"

  # kubectl set image deployment/mydeployment myweb=abhishekraics001/appimage:$image\_version

else

  echo "Deployment does't exit"

kubectl create –f k8/deployment.yaml

}

Note:

* + - 1. grep mydeployment: user to search the deployment or any name
      2. mydeployment: is my deployment name
      3. myweb : this is container name wich mention in deployment file
      4. appimage : image Name
      5. abhishekraics001 : docker hub image name

<https://github.com/devopstraining97/demo-app>

**How to Automatically Trigger the Build in Jenkins**

1. Go To **Configure => Build Trigger** =>
   1. Trigger Build remotely (from script)
   2. Build after other project build
   3. Build periodically
   4. GitHub hook trigger for GITScm Pollling
   5. Poll SCM
2. We are going to learn => **GitHub hook trigger for GITScm Pollling**
3. Go to **github** code repository and create github hook
   1. Go **Git** **Repo**: <https://github.com/abhishekraics001/HTMLStaticApp>
   2. **Setting=> webhooks**: <https://github.com/abhishekraics001/HTMLStaticApp/settings/hooks>
   3. Click on Add **webhooks**
   4. Add your **Payload UR**L: Your running Jenkins url : <http://13.112.77.226:8080/>
   5. Add github-webhook/ after your Jenkins url: : <http://13.112.77.226:8080/github-webhook/>
   6. Change the Application Content Type: **application/json**
   7. Select the Event**: Use the push Event**
   8. **AddHook**

status=$(kubectl get deploy | grep mydeployment | wc -l)

if [ $status -eq 1 ]

then

  echo "Deployment exists in the cluster"

  kubectl set image deployment/mydeployment myweb=abhishekraics001/appimage:latest

  kubectl rollout restart deploy mydeployment

else

  echo "Deployment Doesn't exists"

  kubectl create -f k8/deployment.yaml

  kubectl set image deployment/mydeployment myweb=abhishekraics001/appimage:latest

fi

Note: If you are not able to see/view your image/website/app in browser after build & deploy your docker Image through k8s & Jenkins then got to your master node delete your deployment

* + - * 1. Kubectl get services => Get the list of running services
        2. kubectl get nodes => Get the list of running nodes
        3. kubectl get podes => get the list of running podes
        4. kubectl get deploy => get the list of deployment
        5. kubectl get deployment => get the list of deployment
        6. kubectl delete deploy YourDeploymentName => delete your deployment

**We will learn Maven & java based application deployment**

**Class: 08/12/22 :**

Create build file & push it server like(Project Based on : java, nodejs, angularJS)

**Build tool:** add Dependency, Compile & build the code

1. **Java**: Maven, Ant, Gradle, pom.xml
2. **NodeJS**: NPP, Package.json
3. **Angular, VueJS, ReactJS, NextJS:** NPM, package.json
4. **DotNet:** dotnet, aspx
5. **Python:** pip, requirement.txt

**Maven:**  Build tool for java based application

1. It is build tool for java
2. Is is enforce a directory structure for the java application
3. It provides a easy way to compile, test & package the code

**Artifact**: Output of build is called artifact like .jar & .war

**Maven Phases :**

1. **Validate** : It is used to validate the project to check the project ha necessary information
2. **Compile**: Performed the source code compilation
3. **Test**: Compile code are tested with test phases
4. **Package**: It is going to package the compile the code inform of .jar/.war
5. **Install**: It deploy the package into local repo(.m2 directory)
6. **Deploy**: Push your package into remote maven repository

**Web Server:**

1. For Static Application():
   1. Web Server: **nginx, apach2**
   2. Default Port: **80**
   3. **Nginx** Application file path: **nginx/user/share/nginx/html**
   4. **Apach** Application file path: **/var/www/html**
2. Java Based Web Application:
   1. Web Server: **Apache Tomcat Server**
   2. Default Port: **8080**
   3. **Tomcat** Application file path: **/user/local/tomcate/webapps/**

**How Build the Java-Tomcat based project by Manual**

1. 1st update the apt

sudo apt update -y

1. Install maven in your local system

sudo apt install maven

1. Clone the code

git clone <https://github.com/abhishekraics001/java-demo-tomcat>

1. Check the install version of vaven

mvn --version

1. Build(build, test & package) the code

mvn install

1. Install the code(validate, compile, build, test, package, install)

maven install

1. See the all out(there will be one target folder, .jar or .war file )

**Integrate java-tomcat based application**

1. Pull the java-tomcat base project from git
2. Create & Build Docker image
3. Push the docker image into DockerHub
4. Deployed Docker image into k8 cluster

**Step1:**

1. Install the required software / tool on jenkins server
   1. Maven:
      1. sudo apt update -y
      2. apt install maven -y
2. Install the Jenkins plugin related to that tool: Manage Jenkins > Available Plugins > Maven Integration
   1. Search The maven in jenkin available plugin section and install it (<https://plugins.jenkins.io/maven-plugin/> )
      1. Maven Integration: <https://plugins.jenkins.io/maven-plugin/>
3. Create new Job:
   1. Maven project: You will see this option after installation of “Maven Integration” plugin
   2. JavaProject
   3. Ok
4. Source Code Management
   1. Select the Git:
      1. Repository URL : Add your Java-tomcat base project git Repository

<https://github.com/abhishekraics001/java-demo-tomcat>

* + 1. Credentials: in case of private repo
    2. Branches to build:

Main

1. **Build Triggers**: if you want to trigger build automatically after push code by developer
   * 1. **GitHub hook trigger for GITScm polling**
     2. Go to **github** code repository and create github hook
     3. Go **Git** **Repo**: <https://github.com/abhishekraics001/java-demo-tomcat>
     4. **Setting=> webhooks**: <https://github.com/abhishekraics001/java-demo-tomcat>
     5. Click on Add **webhooks**
     6. Add your **Payload UR**L: Your running Jenkins url(public IP Address & followed by : 8080 ) : <http://13.112.77.226:8080/>
     7. Add github-webhook/ after your Jenkins url: : <http://13.112.77.226:8080/github-webhook/>
     8. Change the Application Content Type: **application/json**
     9. Select the Event**: Use the push Event**
     10. **AddHook**
2. **Build Environment:** 
   1. Pre Step: If you want to add any pre execute any command
   2. Build
      * 1. Root POM: add your root pom path which are in your project
           1. pom.xml
        2. Goals and Options:
           1. install
   3. Post Step: If you want to add any post execute command
      * 1. Select Run only if build succeeds
        2. And Select Docker Build and Push
           1. Add your Docker Repo Details

Repo Name: abhishekraics001/java-tomcat-webapp

Tag: V1

Registry Credentials

1. Build Setp
2. status=$(kubectl get deploy | grep mytomcat-deployment | wc -l)
3. if [ $status -eq 1 ]
4. then
5. echo "Deployment exists in the cluster"
6. kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest
7. kubectl rollout restart deploy mytomcat-deployment
8. else
9. echo "Deployment Doesn't exists"
10. kubectl create -f k8/service.yaml
11. kubectl create -f k8/deployment.yaml
12. kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest
13. fi

Note:

* + 1. **mytomcat-deployment**: Deployment Name
    2. **tomcat**: Container Name
    3. **abhishekraics001**: DockerHub UserName
    4. **java-tomcat-webapp**: DockerHub ImageName
    5. **latest**: it dockerHub image Tag Name

1. Don’t forget to marks as Install automatically for below dependency software
   1. Dashboard => Manage Jenkins => Global Tool Configuration
      1. JDK
      2. Git
      3. Maven
2. Access your App through browser
   1. http://54.64.170.247:30002
   2. <http://54.64.170.247:30002/SimpleTomcatWebApp/>
      1. **54.64.170.247**: Server/EC2 Instance Public IP Address where you have install this application
      2. **30002**: NodePort which is mentioned in your k8/service.yaml file
      3. **SimpleTomcatWebApp**: artifactId(AppName) which is mentioned in your project pom.xml file

**Class: 09/12/22 : CI/CD PipeLine**

**Artifact Server:** Nexus & jFrog => build out package tool

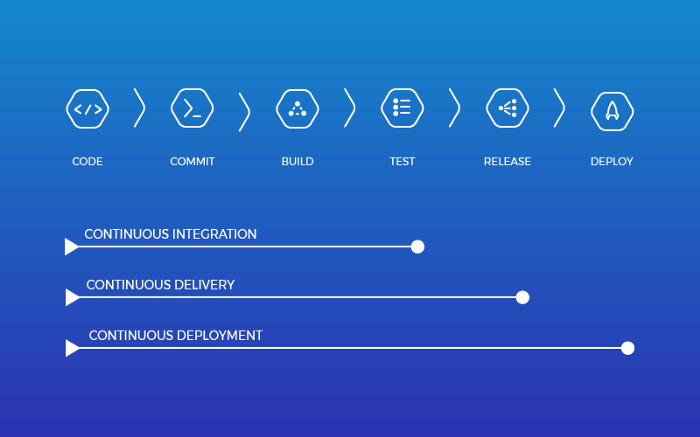
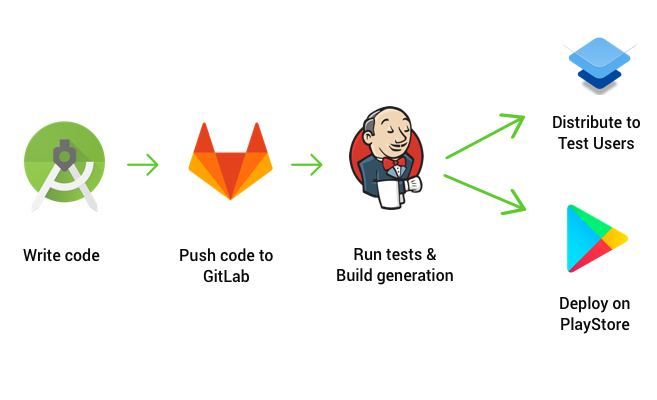
* Jenkins
* Buddy
* TeamCity
* BiG EVAL
* Bamboo
* Integrity
* Wercker
* IBM UrbanCode

**CI/CD:** Continuous Integration & Continuous Delivery => Developer can continuous develops the code and commit to SCM/VCS. And after commit the code automatically build the code and artifact (package the build) the build and deploy it to staging server(not on production/live server)or client/customer

**CI/CD:** Continuous Integration & Continuous Development: Continuous Integration & Continuous Delivery => Developer can continuous develops the code and commit to SCM/VCS. And after commit the code automatically build the code and artifact (package the build) the build and deploy it to staging server as well as Production/live server.

**Continuous Integration (CI)** is the process of automating the build every time a team member commits changes to version control.

**Continuous Deployment (CD)** is a strategy for software releases and CI is a building block of this process.

Integrate CI/CD with Jenkins:

1. Create two replica of job java Web App(JavaProject) which we have previously or create new

**A. Pre-Prod-Job : Staging**

a. Go DashBoard => Add New Items or create a job => Enter your job Name => Pre-Prop-Job

b. Copy From => Enter your previous Job name which you have already created witch like: I want to copy all configuration from our JavaProject Job/Project

c. Change your Deployment Name: mytomcat-stage-deployment

d. Change the branch name(create one more branch on your git repo project with name stag) : stage .

e. Change the deployment Name in your k8 Develoyment.ymal file: mytomcat-stage-deployment

f. Also change the Selector matchLabels Name: tomcat-stage-frontend

g. Also change the template labels Name: tomcat-stage-frontend

h. Change metadata name in service.yaml file : mytomcat-stage-service

i. Change nodePort name in service.yaml file : 30002

status=$(kubectl get deploy | grep mytomcat-stage-deployment | wc -l)

if [ $status -eq 1 ]

then

  echo "Deployment exists in the cluster"

  kubectl set image deployment/mytomcat-stage-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest

  kubectl rollout restart deploy mytomcat-stage-deployment

else

  echo "Deployment Doesn't exists"

  kubectl create -f k8/service.yaml

  kubectl create -f k8/deployment.yaml

  kubectl set image deployment/mytomcat-stage-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest

fi

/K8/deployment.yaml file

apiVersion: apps/v1

kind: Deployment

metadata:

 name: mytomcat-stage-deployment

spec:

 strategy:

   type: RollingUpdate

 replicas: 2

 selector:

   matchLabels:

     app: tomcat-stage-frontend

 template:

   metadata:

    name: myPod01

    labels:

      app: tomcat-stage-frontend

   spec:

    containers:

    - name: tomcat

      imagePullPolicy: Always

      image: abhishekraics001/java-tomcat-webapp:v1

      ports:

        - containerPort: 8080

/k8/service.yaml file

apiVersion: v1

kind: Service

metadata:

  name: mytomcat-stage-service

spec:

  type: NodePort

  ports:

    - port: 8080 # port on which clusterIP will listen on

      targetPort: 8080 # container Port

      nodePort: 30004

  selector:

    app: tomcat-frontend

**B. Prod-Job : Production**

a. Go DashBoard => Add New Items or create a job => Enter your job Name => Prop-Job

b. Copy From => Enter your previous Job name which you have already created witch like: I want to copy all configuration from our JavaProject Job/Project

c. Change your Deployment Name or if you want same then you can also keep it same: mytomcat-prod-deployment Or mytomcat-deployment

d. Change the branch name(create one more branch on your git repo project with name stag) : main .

e. unselect your Build Trigger : GitHub hook trigger for GITScm Polling

status=$(kubectl get deploy | grep mytomcat-deployment | wc -l)

if [ $status -eq 1 ]

then

  echo "Deployment exists in the cluster"

  kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest

  kubectl rollout restart deploy mytomcat-deployment

else

  echo "Deployment Doesn't exists"

  kubectl create -f k8/service.yaml

  kubectl create -f k8/deployment.yaml

  kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest

fi

/K8/deployment.yaml file

apiVersion: apps/v1

kind: Deployment

metadata:

 name: mytomcat-deployment

spec:

 strategy:

   type: RollingUpdate

 replicas: 2

 selector:

   matchLabels:

     app: tomcat-frontend

 template:

   metadata:

    name: myPod01

    labels:

      app: tomcat-frontend

   spec:

    containers:

    - name: tomcat

      imagePullPolicy: Always

      image: abhishekraics001/java-tomcat-webapp:v1

      ports:

        - containerPort: 8080

/k8/service.yaml file

apiVersion: v1

kind: Service

metadata:

  name: mytomcat-service

spec:

  type: NodePort

  ports:

    - port: 8080 # port on which clusterIP will listen on

      targetPort: 8080 # container Port

      nodePort: 30001

  selector:

    app: tomcat-frontend

**Docker File for both**

FROM tomcat

COPY target/\*.war /usr/local/tomcat/webapps/

**Install CI/CD Pipeline:**

* + - 1. Dashboard => manage jenkins => plugin manager => available plugin => Search “Build Pipeline”

<https://plugins.jenkins.io/build-pipeline-plugin/>

* + - 1. **Post Build Action for CD Part**:
         1. Post Build Action and Select the “**Build Others Project**” and

Add your Project JobName in **Project To Build**

And Select “Trigger only if Build is stable**”**

* + - * 1. Post Build Actions and Select the **“Build Other Project(Menual Options)”** and this options will be available after install the **“Build Pipeline”**

Add You Downstream Project Job Name like: Prod-job

Prod-job => Your job name (which will run after/post of current project or job)

* + - 1. **Create a New View (+) for visualizing the pipeline** 
         1. Name: CI-CD-View
         2. Type: Build Pipeline View
         3. Pipeline Flow: => Upstream / Downstream config

Select your initial pipeline : Pre-Prod-Job

No of Build to Dispay as your choice: like 5

* + - 1. Add Extra condition in Executable Shell for check device deployment of not

#To check mytomcat-deployment is exit or not  
status=$(kubectl get deploy | grep mytomcat-deployment | wc -l)  
if [ $status -eq 1 ]  
then  
 echo "Deployment exists in the cluster"  
 kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest  
 kubectl rollout restart deploy mytomcat-deployment  
  
 #To check SimpleTomcatWebApp is running or not  
 status=$(curl -I http://43.206.107.118:30007/SimpleTomcatWebApp/ | grep -we 200 | wc -l)  
 if [ $status -eq 1 ]  
 then  
 echo "Application is up"  
 else  
 echo "Application is up is not giving expected erro"  
 kubectl rollback deploy mytomcat-deployment  
 fi  
else  
 echo "Deployment Doesn't exists"  
 kubectl create -f k8/service.yaml  
 kubectl create -f k8/deployment.yaml  
 kubectl set image deployment/mytomcat-deployment tomcat=abhishekraics001/java-tomcat-webapp:latest  
fi

**Jenkins Master / Slave Arch:**

1. Create one EC2 instance as master node: We we install Jenkins UI, Create & Manage Jenkins job
2. Create 1 more EC2 instance as Slaves/Worked node & setup Jenkins on salves node:
   1. Install Java on Slave Node:
      1. sudo apt update –y
      2. sudo apt install openjdk-11-jre -y
   2. Jenkins user and Jenkins Home
      1. Jenkins Home(like current user is ubuntu or abhirai):
      2. Add Jenkins user:
         1. Pwd => will print (/home/loginUserName)
3. Go to Jenkins through master node => Manage Jenkins => Manage Nodes and Clouds
   1. Click on Add Node (+) =>
   2. Node Name: Slave1
   3. Select Permanent Agent
   4. Create
   5. Add No of executer like => 4
   6. Remote root directory (home directory of your slave node) and path is used by master node to execute job on Slave nodes: /home/ubuntu/jenkins
   7. Label: like Tag name which is used to as Tag: myslave-on-aws
   8. Lunch Method:
      1. Select Lunch Agent Via SSH
      2. IP Address of Slave Node: 54.248.173.95
      3. Credentials: => Add Users
         1. Kind: SSH username and password
         2. Scope: Global(Jenkins, nodes, etc)
         3. ID(Enetr any): slave-ssh-key
         4. Username(SSH user name, Slave node machine username): ubuntu
         5. Private Key
            1. Enter Directory :
            2. Add(Add your Private key => Download your .pem file from aws by which you created your slave node ec2 machine and open it in text editor and copy all content pest it inputbox ):
            3. Add
      4. After adding Select your credentials
      5. Host Key Verification Strategy : Non verifying verifition strategy
4. After That **Select/click on your newly created node** and **Connect**
5. After Go To Your Job
   1. Configure
      1. Select/Check the: Restrict where this project can be run
         1. Search/Add your slave node label
6. After that go Slave node install git & docker
   1. sudo apt update –y
   2. sudo apt install git –y
   3. sudo apt install docker.io –y
   4. sudo usermod ubuntu -G docker
   5. exit
   6. Again login to Slave node & run below command
   7. Then **Install Kubectl**

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg <https://packages.cloud.google.com/apt/doc/apt-key.gpg>

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update –y

sudo apt install kubectl –y

* 1. Go to your k8s master node and copy your cubeconfig file (private & public key) and add it in your salve node by which your jenkins slave node communicate with your master node
     1. Master node
        1. cd
        2. ls –a
        3. cd .kube/
        4. ls
        5. cat config
        6. copy all content wich are displaying after hi cat command
     2. Go to your jenkin slave node
        1. create .kube directory with your user(ubuntu)
           1. mkdir .kube
           2. cd .kube/
           3. vim config
           4. And pest your master node private & public key
           5. ESC
           6. :wq
     3. After that run this command in jenkin slave node
        1. kubectl get pods

1. After go to your jenkin job and select the
   1. Restrict where this project can be run
      1. Add your slave node name : myslave-on-aws

Android project Build Automations:

Help Link: <https://bugfender.com/blog/how-to-add-your-first-android-job-to-jenkins/>

* + - 1. Create & Setup AWS EC2 Instance
      2. Install & setup Java & Jenkins
      3. Install unziper for unzip the folder
         1. sudo apt-get install unzip
      4. Download & Install Android sdk-tools for linux
         1. sudo -iu jenkins wget <https://dl.google.com/android/repository/sdk-tools-linux-3859397.zip>
         2. sudo -iu jenkins mkdir android-sdk
         3. sudo -iu jenkins unzip sdk-tools-linux-3859397.zip -d android-sdk