

JENKINS

CONTINUOUS INTEGRATION: It is the combination of continuous build and continuous test.

CONTINUOUS INTEGRATION = CONTINUOUS BUILD + CONTINUOUS TEST

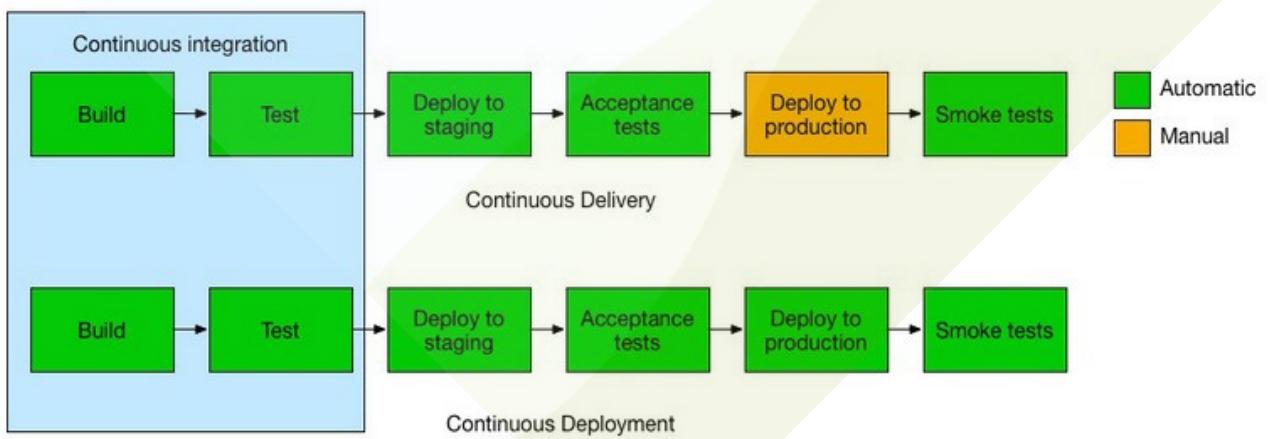
Whenever a developer commits the code using source code management like GIT, then the CI pipeline gets the changes of the code runs automatically build and unit test.

- Due to integrating the new code with old code, we can easily get to know the code is a success or failure.
- It finds the errors more quickly
- Delivers the products to client more frequently
- Developers don't need to manual tasks
- Reduces the developers time 20% to 30%

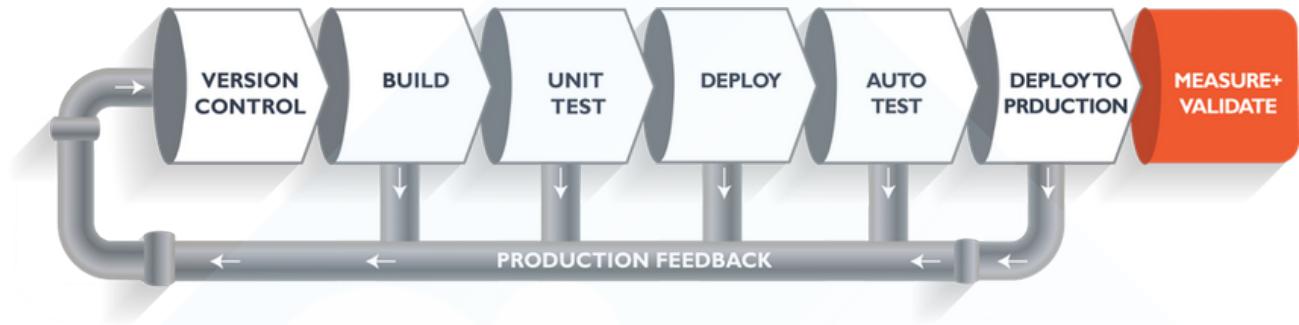
CD: CONTINUOUS DELIVERY/DEPLOYMENT:

Continuous Delivery is making it available for deployment. Anytime a new build artifact is available, the artifact is automatically placed in the desired environment and deployed.

Continuous Deployment is when you commit your code then it gets automatically tested, build and deploy on the production server.



CI/CD PIPELINE: A CI/CD pipeline is a set of automated steps and tasks that code changes go through, from the initial commit to deployment. These pipelines are typically defined and configured using CI/CD tools and scripts. A pipeline can include actions such as code compilation, testing, packaging, and deployment.



It looks like Software Development Life Cycle, but let's see how it works. Lets consider an example, if you are developing a web application.

Version Control: Here developers need to write code for web applications. So it needs to be committed using version control system like GIT or SVN.

Build: Lets consider your code is written in java, it needs to be compiled before execution. In this build step code gets compiled.

Unit Test: If the build step is completed, then move to testing phase in this step unit step will be done.

Deploy: If the test step is completed, then move to Deploy phase in this step you can deploy your code in dev, testing environment. Here you can see your application output.

Auto Test: Once our code is working fine in testing servers, then we need to do Automation testing using Selenium or Junit.

Deploy to Production: If everything is fine then you can directly deploy your code in production server.

NOTE: If we have error in Code then it will give feedback and it will be corrected, if we have error in Build then it will give feedback and it will be corrected, Pipeline will work like this until it reaches Deploy.

Because of this pipeline, Bugs will be reported fast and get rectified so entire development is fast .

JENKINS:

- Jenkins is an **open source** project written in **java** by Kohsuke Kawaguchi it runs on the **Window, Linux and Mac OS .**
- Jenkins is a widely used open-source automation tool that helps developers and teams automate and streamline various tasks related to building, testing, and deploying the

- applications
- It is community-supported, Free to use, and the First choice for Continuous Integration.
- Consist of Plugins
- Automates the Entire Software Development Life Cycle (SDLC).
- It was originally developed by Sun Microsystem in 2004 as HUDSON.
- Hudson was an enterprise Edition we need to pay for it.
- The project was renamed Jenkins when Oracle brought the Microsystems. It can run on any major platform without Compatibility issues.
- Whenever developers write code, we integrate all the code of all developers at any point in time and we build, test, and deliver/deploy it to the client. This is called CI/CD.

Advantages:

- Jenkins follows Master-Slave Architecture.
- You can write your own plugin, can use the community plugin also.
- Jenkins master is going to assign a job to the slave..
- If Slaves are not available Jenkins itself does the job.
- By using the labels we can specify the jobs to the nodes.
- Jenkins automates repetitive and manual tasks in the software development process, reducing human error and saving time.
- Jenkins enables teams to continuously integrate code changes, making it easier to detect and fix integration issues early in the development cycle.
- Jenkins can be configured for high availability, ensuring continuous operation and minimal downtime.
- It integrates seamlessly with popular version control systems like Git, enabling efficient code management.

JENKINS ALTERNATIVES:

- BAMBOO
- HUSON
- TEAM CITY
- CIRCLE CI
- AWS CODE PIPELINE
- BUDDY
- SEMAPHORE
- GITLAB

JENKINS SETUP:

STEP-1: LAUNCH AN EC2 INSTANCE WITH SEPARATE SECURITY GROUP

STEP-2: GO TO JENKINS.IO AND COPY PASTE THOSE 2 LINKS

STEP-3: INSTALL JAVA

STEP-4: INSTALL JENKINS

STEP-5: START THE JENKINS

links-1: `sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo`

link-2 : `sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key`

To install java11 : `amazon-linux-extras install java-openjdk11 -y`

To install jenkins : `yum install jenkins -y`

To start jenkins : `systemctl start jenkins`

To check jenkins status : `systemctl status jenkins`

To stop jenkins : `systemctl stop jenkins`

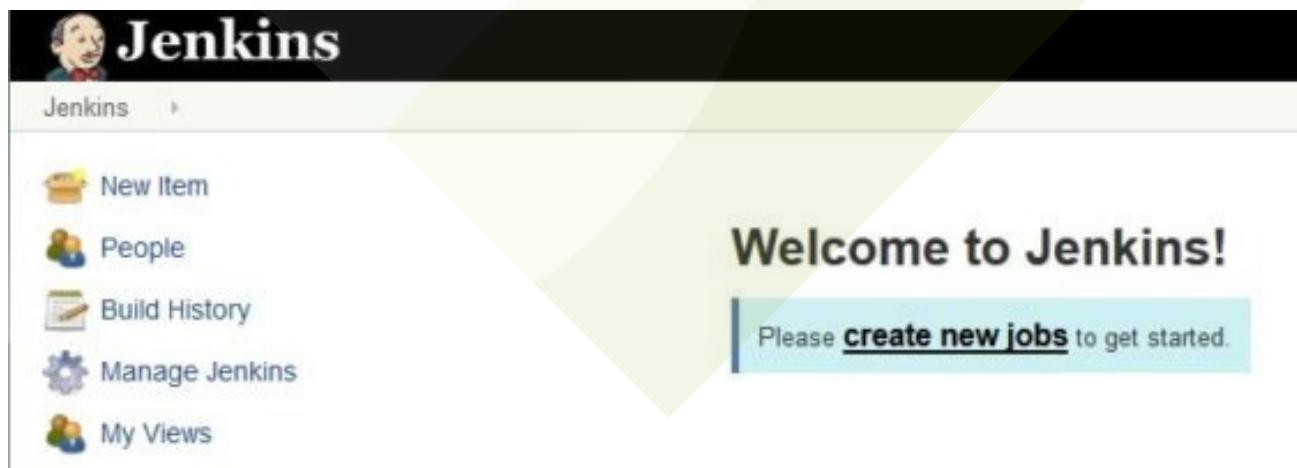
Connect to dashboard: copy the public IP address of the server and make a paste on the new tab with Jenkins port number (8080)

`public_ip:8080`

To unlock the jenkins:

`cat /var/lib/jenkins/secrets/initialAdminPassword`

- Install suggested plugins
- Add user name, Password and mail id
- connect to dashboard





Credentials



New View

JOB: To perform some set of tasks we use a job in jenkins.

- A Jenkins job is like a task or set of instructions that you create and configure in Jenkins to automate a specific action in your software development process.
- These actions can include things like compiling code, running tests, or deploying software.
- Jenkins jobs are used to define and manage the automation of these tasks, making it easier for developers to consistently and reliably execute them.



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

PARAMETER TYPES:

- String: any combination of characters and numbers
- Choice: a pre-defined set of strings from which a user can pick a value
- Credentials: a pre-defined Jenkins credential
- File: the full path to a file on the filesystem
- Multi-line String: same as String, but allows newline characters

- Password: similar to the Credentials type, but allows us to pass a plain text parameter specific to the job or pipeline
- Run: an absolute URL to a single run of another job

FILE PARAMETER:

This is used when we want to build our local files.

General –> This Project is Parameterized –> File Parameter

CHOICE PARAMETER:

This parameter is used when we have multiple options to generate a build but need to use only one specific one.

General –> This Project is Parameterized –> Choice Parameter

STRING PARAMETER:

This parameter is used when we need to pass a parameter as input by default.

It can be any combination of characters and numbers.

General –> This Project is Parameterized –> String Parameter

MULTI STRING PARAMETER:

This will work as same as String Parameter but the difference is instead of one single line string we can use multiple strings at a time as a Parameters.

General –> This Project is Parameterized –> Multi-String Parameter

LINKED JOBS :

This is used when a job is linked with another job

TYPES:

Up stream and Down stream

Here for job-1 both job-2 & job-3 are Downstream

For job-2 upstream is job-1 and Downstream is job-3

for Job-3 Both Job-2 & job-1 are Upstream



JENKINS MASTER & SLAVE:

Jenkins Master-Slave architecture is a setup where you have one main Jenkins server (the "Master") that manages and controls multiple additional servers (the "Slaves").

- The Master server coordinates and schedules tasks.
- The Slave servers do the actual work, like building, testing, and deploying applications.

This architecture allows you to distribute the workload and run tasks in parallel, making your Jenkins setup more efficient and scalable. It's like having a boss (Master) who delegates tasks to a group of workers (Slaves) to get the job done faster.

STEPS TO IMPLEMENT JENKINS MASTER-SLAVE:

1. LAUNCH 3 INSTANCES WITH KEYPAIR (PEM FILE)

- MASTER
- SLAVE-1
- SLAVE-3

2. SETUP JENKINS IN MASTER

- install **JAVA11** on slaves (amazon-linux-extras install java-openjdk11 -y)
- go to manage jenkins >> manage nodes and clouds >> new node --> give node name & select permanent agent

New Node	
Name	slave-1
Description	this is for slave-1
Number of executors	3
Remote root directory	/home/ec2-user/jenkins-slave

Labels: dev

Usage: Only build jobs with label expressions matching this node

Launch method: Launch agents via SSH

Host: 172.31.44.108

ADD CREDENTIALS:

Add Credentials

Domain: Global credentials (unrestricted)

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

ID: (empty)

Description: this is for slave-2

Username: ec2-user

Description: this is for slave-2

Username: ec2-user

Treat username as secret

Private Key: Enter directly

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQDwP1iu182qOpTEnxt28cfayLan39ccsdXIVRfqTx3oxw0lfv05T5Uc5xF80
UltaRtvch4Whn0Phr+1jO4rsb8kXTX4blePzavkywoq+9z7n01x0
-----END RSA PRIVATE KEY-----
```

Passphrase: (empty)

Add **Cancel**

Disable deferred wipeout on this node

Environment variables

Host: 172.31.44.108

Credentials: ec2-user (this is for slave-1)

+ Add

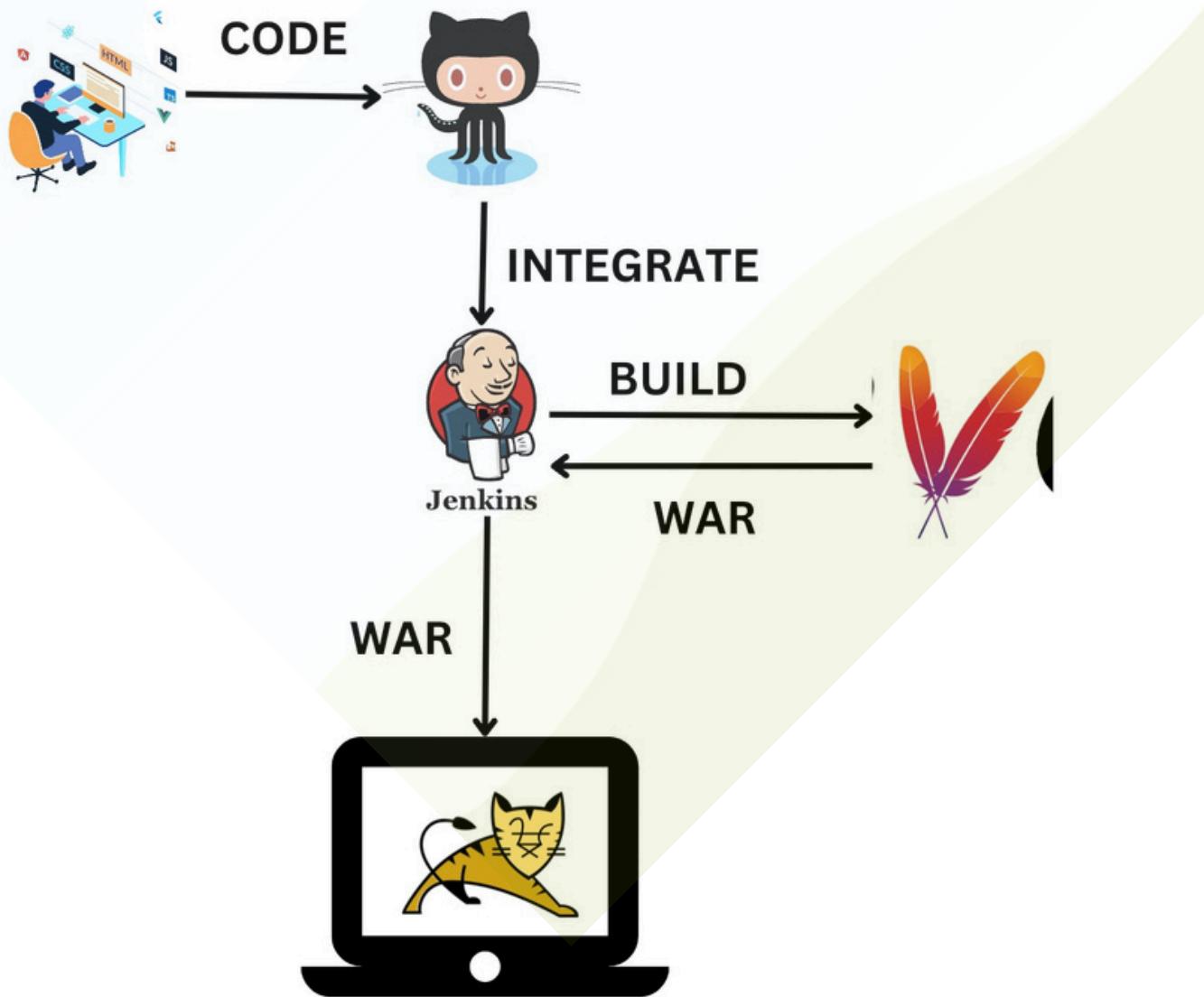
Host Key Verification Strategy: Non verifying Verification Strategy

The screenshot shows the Jenkins Node Properties configuration page. At the top, there is a "Save" button and a "Advanced..." link. Below that, the "Availability" section contains a dropdown menu with the option "Keep this agent online as much as possible". Under "Node Properties", there are three checkboxes: "Disable deferred wipeout on this node", "Environment variables", and "Tool Locations". A "Save" button is located at the bottom of this section.

CLICK ON SAVE

JENKINS PROJECT:

DEPLOY A WAR FILE IN TOMCAT SERVER:



REQUIREMENTS:

- LAUNCH 2 SERVER (jenkins, prod)
- GETTHECODE FROMTHE DEVELOPERS
- SETUP JENKINS IN JENKINS SERVER
- SETUP TOMCAT IN PROD SERVER

PROCEDURE:

STEP-1: LAUNCH 2 INSTANCES WITH 8080 PORT

STEP-2: SETUP JENKINS IN SERVER

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

```
amazon-linux-extras install java-openjdk11 -y
```

```
yum install jenkins -y
```

```
systemctl restart jenkins
```

STEP-3: FORK THE GITHUB (<https://github.com/devops0014/one.git>)

STEP-4: INSTALL GIT IN OUR SERVER

```
yum install git -y
```

STEP-5:

CREATE JOB AND INTEGRATE GIT TO JENKINS AND BUILD IT.

ONCE WE BUILD THE JOB, FILES PRESENT IN MASTER BRANCH WILL COMES INTO CI SERVER

STEP-6: NEXT STEP IS BUILD THE SOURCE CODE WHICH ARE PRESENT IN CI SERVER. TO BUILD THE WE NEED TO USE MAVEN

INSTALL JAVA-1.8.0 & MAVEN IN OUR SERVER

```
yum install java-1.8.0-openjdk -y
```

```
yum install maven -y
```

STEP-7:

**CONFIGURE THE SAME JOB AND CLICK ON BUILD STEP AND SELECT ADD BUILD STEP
SELECT invoke top level maven target.
in the goal : clean package**

SAVE THE JOB AND BUILD.

SO WE WILL GET A WAR FILE IN TARGET FOLDER.

STEP-8: SETUP THE TOMCAT SERVER IN PROD SERVER.

1. download tomcat file from dlcdn: [wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.70/bin/apache-tomcat-9.0.70.tar.gz](https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.70/bin/apache-tomcat-9.0.70.tar.gz)
2. untar the file: [tar -zvxf apache-tomcat-9.0.70.tar.gz](#)
3. go to the folder: [cd apache-tomcat-9.0.70//webapps/manager/META-INF](#)
4. open the context.xml in vim editor and make some change (delete 2 lines (21 and 22 lines))
5. go to three steps back: [cd ../../..](#)
6. and go to conf folder and open tomcat-user.xml file in vim editor

```
-->
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="tomcat" password="123456" roles="manager-gui, manager-script"/>
</tomcat-users>
```

7. go to one step back: [cd ..](#)

8. go to bin folder and execute startup.sh file

9. [./startup.sh](#)

STEP-9: Go to manager apps and it will ask the user name and password enter it

STEP-10: go to jenkins dashboard

- install plugin (manage jenkins --> manage plugin --> available plugin --> [deploy to container](#))
- after installing the plugin go to our job and select post build actions --> add post build actions.
- select deploy war/ear to container

Post-build Actions

Deploy war/jar to a container

WAR/LAR files

Context path

Containers

Deploy on failure

- click on add container(9th version). and add credentials (username & password of tomcat)

Add Credentials

Domain

Kind

Scope

Username

Treat username as secret

Password

ID

- add tomcat url

Tomcat URL

Deploy on failure

- save and build the job and go to tomcat

you will see swiggy folder. click on the folder you can access the client application.

Jenkins User Management:

It is the process of controlling and organizing the individuals who can access and use Jenkins, the automation tool. It involves tasks like creating user accounts, setting permissions, and managing who can do what within Jenkins. User management ensures that the right people have the appropriate level of access and control over Jenkins while maintaining security and organization in the software development process.

- INSTALL PLUGIN (role-based authorization strategy). INSTALL WITHOUT RESTART & RESTART JENKINS
- go to manage jenkins and select manage users.
- create user

The screenshot shows the Jenkins 'Create User' interface. At the top, there's a navigation bar with links for 'Dashboard', 'Jenkins' own user database', and 'Create User'. The main area is titled 'Create User'. It contains five input fields: 'Username' (akshay), 'Password' (redacted), 'Confirm password' (redacted), 'Full name' (akshay bhole), and 'E-mail address' (akshay@infosys.com). Below these fields is a blue 'Create User' button.

- 4. Go to manage jenkins >> config global security and change the authorization to role based strategy and save it

The screenshot shows the Jenkins 'Configure Global Security' interface. At the top, there's a navigation bar with links for 'Dashboard', 'Manage Jenkins', and 'Configure Global Security'. The main area is titled 'Configure Global Security'. It has three main sections: 'Authentication' (with 'Disable remember me' checked, 'Jenkins' own user database' selected, and 'Allow users to sign up' checked), 'Authorization' (with 'Role-Based Strategy' selected), and 'Markup Formatter' (with 'Plain text' selected). A note at the bottom states: 'Treats all input as plain text. HTML, unsafe characters like < and & are escaped to their respective character entities.'

Amount

- Go to manage jenkins >> manage and assign roles and select manage roles.

The screenshot shows the Jenkins 'Manage Roles' page. On the left, there's a sidebar with links like '+ New Item', 'People', 'Build History', etc. The main area is titled 'Manage Roles' and shows a 'Global roles' section. A large grid table lists roles (admin, dev, devops, test) against various Jenkins features (Overall, Credentials, Agent, Job, Run, View, SCM). Each feature has several checkboxes representing different permissions, with some checked for certain roles.

- Go to assign roles

The screenshot shows the Jenkins 'Assign Roles' page. It has a similar sidebar to the previous page. The main area is titled 'Assign Roles' and shows a 'Global roles' section. A grid table lists users/groups (akshey bhalne, gunu kiran, prajwal isane, shakir mustafa, Anonymous) against roles (User/group, Admin, Devops, Test). Checkmarks indicate which roles are assigned to each user/group.

JENKINS BACKUP:

Whatever we execute via Jenkins or perform any setting in Jenkins, it stores in the form of backup in Jenkins home directory. This backup includes data and configuration-related settings like job configurations and plugins, plugin configurations, build logs, etc. So, it's essential to take the backup of this prominent information to use this backup in the future if required.

- INSTALL PLUGIN (THINBACKUP).
- create a folder in opt directory: `mkdir /opt/mybackup`
- change owners of the folder: `chown jenkins:jenkins /opt/mybackup`
- give full permissions to the folder: `chmod 777 /opt/mybackup`
- go to manage jenkins and select thinbackup and click on settings.

The screenshot shows the Jenkins 'ThinBackup Configuration' page. It has a sidebar with links like '+ New Item', 'People', 'Build History', etc. The main area is titled 'ThinBackup Configuration' and shows a 'Thin Backup settings' section. There's a 'Backup directory' input field containing the path `/opt/jenkins/mybackup`.

Check File Fingerprint

Manage Jenkins

My Views

Build Queue
No builds in the queue.

Build Executor Status

- Built-in Node
 - 1 idle
 - 2 idle
 - slave-1
 - 1 idle
 - 2 idle
 - slave-2
 - 1 idle
 - 2 idle
- slavekeypair.pem

Backup schedule for full backups

Backup schedule for differential backups

Max number of backup sets: -1

Files excluded from backup (regular expression):

Wait until Jenkins is idle to perform a backup

Force Jenkins to quiet mode after specified minutes: 120

Backup build results

- Backup build archive
- Backup only builds marked to keep

Backup userContent folder

Backup next build number file

Backup plugins archives

Backup config-history folder

Backup additional files

Clean up differential backups

Move old backups to ZIP files

Show all

Dashboard > Manage Jenkins > ThinBackup

Build Executor Status

- Built-in Node
 - 1 idle
 - 2 idle
 - slave-1
 - 1 idle
 - 2 idle
 - slave-2
 - 1 idle
 - 2 idle
- slavekeypair.pem

Max number of backup sets: -1

Files excluded from backup (regular expression):

Wait until Jenkins is idle to perform a backup

Force Jenkins to quiet mode after specified minutes: 120

Backup build results

- Backup build archive
- Backup only builds marked to keep

Backup userContent folder

Backup next build number file

Backup plugins archives

Backup config-history folder

Backup additional files

Clean up differential backups

Move old backups to ZIP files

Save

Jenkins 2.375.2

slavekeypair.pem

Show all

- click on **backup now**
- go and check in **/opt/mybackup folder**, you will get all the config files of jenkins in our folder.

if you want to restore those files, click on restore option

Jenkins

Dashboard > ThinBackup

ThinBackup

+ New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

Backup now

Restore

Settings

Search (⌘+K)

shaik mustafa

log out

WHAT IS JENKINS FILE?

If we are writing the entire jenkins pipeline in a text format. It contains the steps that are required for running the jenkins pipeline.

It is used to create a pipeline for **build and deploy** the code. This Jenkins file uses **groovy syntax**.

These jenkins file will written in 2 ways

1. Declarative pipeline
2. Scripted Pipeline

Declarative pipeline: It is a **recent feature** of the jenkins pipeline which helps us to write the pipeline in a easier way. It will starts with the word **pipeline**

Scripted pipeline: It is a **traditional way** of writing a jenkins pipeline as a code. It starts with the word **node**

SINGLE STAGE PIPELINE:

```
pipeline {  
  
    agent any  
  
    stages {  
  
        stage("stage-1") {  
  
            steps {  
  
                echo "hai this is my first stage"  
  
            }  
  
        }  
  
    }  
}
```

pipeline : A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

agent : it defines that, in which server the pipeline will gets executes.

stage : stage in jenkins file contains of unique tasks such as build, test, Deploy etc..

step : It tells the jenkins what exactly needs to be done.

ex: executing a build command or linux command etc..

MULTI-STAGE PIPELINE:

```
pipeline {  
    agent any  
    stages {  
        stage ("MUSTAGA") {  
            steps {  
                echo "hello Mustafa"  
            }  
        }  
        stage ("d e v o p s ") {  
            steps {  
                echo "we are learning devops"  
            }  
        }  
        stage ("a w s ") {  
            steps {  
                echo "we are learning aws also"  
            }  
        }  
    } }
```

PIPELINE AS A CODE: YOU CAN RUN COMMANDS HERE

```
pipeline {  
    agent any  
    stages {  
        stage('CMD') {  
            steps {  
                sh 'touch file1'  
                sh 'pwd'  
            }  
        }  
    }  
}
```

MULTIPLE COMMANDS OVER SINGLE LINE:

```
pipeline {  
    agent any  
    stages {  
        stage('CMD') {  
            steps {  
                sh ""  
                touch file2  
                pwd  
                date  
                whoami  
                ""  
            }  
        }  
    }  
}
```

ENVIRONMENT VARIABLES:

```
pipeline {  
    agent any  
    environment {  
        name = 'raham'  
    }  
    stages {  
        stage('ENV') {  
            steps {  
                echo "hai my name is $name"  
            }  
        }  
    }  
}  
  
pipeline {  
    agent any  
    environment {  
        name = 'raham'  
    }  
    stages {  
        stage('ENV1') {  
            steps {  
                sh'echo "${name}"'  
            }  
        }  
        stage('ENV2') {  
            environment {  
                name = 'shaik'  
            }  
            steps {  
                sh'echo "${name}"'  
            }  
        }  
    }  
}
```

```
        sh 'echo "${name}"'  
    }  
}  
}  
}
```

PIPELINE TO GET A SOURCE CODE TO DEV SERVER:

```
pipeline {  
    agent {  
        node {  
            label'dev'  
        }  
    }  
    stages {  
  
        stage("git") {  
            steps {  
                git "https://github.com/devops0014/devoprepoforpractice.git"  
            }  
        }  
    }  
}
```

PIPELINE WITH PARAMETERS:

STRING:

```
pipeline {  
    agent any  
    parameters {
```

```
        string (name: "aws", defaultValue: "EC2", description: "i am unsing aws cloud")
    }
    stages {
        stage ("stage-1") {
            steps {
                echo "hai i am using parameters"
            }
        }
    }
}
```

BOOLEAN:

```
pipeline {
    agent any
    parameters {
        booleanParam (name: "jenkins", defaultValue: true, description: "")
    }

    stages {
        stage('Hello') {
            steps {
                echo ' i am using boolean parameter'
            }
        }
    }
}
```

CHOICE:

```
pipeline {
    agent any
    parameters {
```

```
choice(name: "branch", choices : ["one", "two", "three", "four"], description: "this is cp")
}

stages {
    stage('Hello') {
        steps {
            echo 'Hello World'
        }
    }
}
```

PIPELINE WITH POST BUILD ACTIONS:

ALWAYS:

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World
            }
        }
    }

    post{
        always {
            echo 'THIS WILL BE PRINTED ANYWAY'
        }
    }
}
```

SUCCESS:

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World  
            }  
        }  
    }  
  
    post{  
        success {  
            echo 'THIS WILL BE PRINTED IF THE BUILD GETS SUCCESS'  
        }  
    }  
}
```

FAILURE:

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World  
            }  
        }  
    }  
  
    post{  
        failure {  
    }
```

```
        echo 'THIS WILL BE PRINTED EVEN IF THE BUILD GETS FAILURE'  
    }  
}  
}
```

SUCCESS, ALWAYS, FAILURE:

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
    }  
    post {  
        always {  
            echo "this will gets printed anyway"  
        }  
        success {  
            echo "this will be printed if the build gets success"  
        }  
        failure {  
            echo "this will be printed even if the build gets failure"  
        }  
    }  
}
```

INPUT IN JENKINS FILE:

```
pipeline {  
    agent any  
  
    stages {  
        stage('deploy') {  
            input {  
                message "can i deploy"  
                ok "yes you can"  
            }  
            steps {  
                echo 'our code is deployed'  
            }  
        }  
    }  
}
```

JENKINS FILE TO DEPLOY THE CODE IN TOMCAT SERVER:

```
pipeline {  
    agent any  
    stages {  
        stage ("code") {  
            steps {  
                git "https://github.com/devops0014/myweb-01.git"  
            }  
        }  
        stage ("build") {  
            steps {  
                sh 'mvn clean package'  
            }  
        }  
    }  
}
```

```
        }
    }
    steps {
        sh "cp /var/lib/jenkins/workspace/pipeline-1/target/*.war /opt/apache-tomcat-9.0.71/webapps"
    }
}
}
```

CHANGE TOMCAT PORT NUMBER:

- To stop the tomcat: `./shutdown.sh`
 - Go to conf folder and open server.xml file in vim editor
 - Change the port number in 69 line from 8080 to 8081
 - and save & exit from the file
 - Go to one step back `cd ..`
 - Go to bin folder again and execute `startup.sh` file

JENKINS JOB TO DEPLOY A WEBAPP IN CLIENT SERVER:

1. LAUNCH AN TWO INSTANCES WITH **8080 PORT** AND PEM FILE. (JENKINS & PROD SERVER)
 2. SETUP **JENKINS IN JENKINS SERVER**
 3. INSTALL **JAVA-11 IN PROD SERVER**
 4. CEATE A NODE : go to manage jenkins >> manage nodes and clouds >> new node --> give node name & select permanent agent.



The screenshot shows the Jenkins Credentials Provider dialog for adding credentials. The 'Kind' is set to 'SSH Username with private key'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'Username' is 'ec2-user'. There are two unchecked checkboxes at the bottom: 'Disable deferred wipeout on this node' and 'Environment variables'.

Host: 172.31.44.108

Credentials: ec2-user (this is for slave-1)

Host Key Verification Strategy: Non verifying Verification Strategy

Availability: Keep this agent online as much as possible

Node Properties

- Disable deferred wipeout on this node
- Environment variables
- Tool Locations

Save

REST API Jenkins 2.375.2

- SETUP THE TOMCAT IN PROD SERVER (dont need to change the port number).
- INSTALL PLUGIN: go to manage jenkins >> manage plugins >> available plugin >> deploy to container.

CREATE A JOB IN FREE STLE:

The screenshot shows the Jenkins Configuration screen with the 'General' tab selected. The 'Enabled' checkbox is checked.

Dashboard > MyApp > Configuration

Configure **General**

Enabled

General

Description

[Plain text] Preview

Discard old builds

GitHub project

Project url: <https://github.com/devops0014/myweb-01.git>

This project is parameterised

Throttle builds

Execute concurrent builds if necessary

Restrict where this project can be run

Label Expression: prod

Label prod matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

Dashboard > MyApp > Configuration

Configure

Source Code Management

None

Git

Repositories

Repository URL: <https://github.com/devops0014/myweb-01.git>

Credentials: - none -

+ Add

Add Repository

Branches to build

Branch Specifier (Blank for "any")

*/*master

Add Branch

Repository browser

Dashboard > MyApp > Configuration

Configure

Build Steps

Invoke top-level Maven targets

Goals: clean package

Dashboard > MyApp > Configuration

Configure

Post-build Actions

Deploy war/jar to a container

WAR/EAR files: target/*.war

Context path: swiggy

Containers

Tomcat 9.x Remote

Credentials: tomcat*****



SAVE & BUILD

SOME ADDITIONAL TOPICS:

JENKINS POST BUILD ACTIONS:

```
pipeline {
```

```
    agent any
```

```
    stages {
```

```
        stage('Hello') {
```

```
            steps {
```

```
                echo 'Hello World'
```

```
            }
```

```
            post {
```

```
                always {
```

```
                    echo "this will be printed anyways"
```

```
                }
```

```
            }
```

```
        }
```

```
        post {
```

```
always {
    echo "This block always runs."
}

changed  {
    echo "This block runs when the current status is different than the previous one."
}

fixed   {
    echo "This block runs when the current status is success and the previous one was failed or unstable."
}

regression  {
    echo "This block runs when the current status is anything except success but the previous one was successful."
}

unstable  {
    echo "This block runs if the current status is marked unstable."
}

aborted  {
    echo "This block runs when the build process is aborted."
}

failure  {
    echo "This block runs when the build is failed."
}

success   {
    echo "This block runs when the build is succeeded."
}

unsuccessful  {
```

```
        echo "This block runs when the current status is anything except success."  
    }  
}  
}
```

JENKINS WHEN CONDITION:

```
pipeline {  
    agent any  
    stages {  
        stage("Test") {  
            when {  
                equals(actual: currentBuild.number, expected: 5)  
            }  
            steps {  
                echo "Hello World!"  
            }  
        }  
    }  
}
```

Note: In the above pipeline, when the build id is 5, then only stage will gets executed.

WHEN NOT:

```
pipeline {
```

```
agent any

stages {

    stage("Test") {

        when {
            not {
                branch "master"
            }
        }

        steps {
            echo "Test stage."
        }
    }
}
```

ALL OF:

```
pipeline {

    agent any

    stages {

        stage("Deploy") {

            when {
                allOf {
                    branch "master"
                    environment(name: "ENV", value: "production")
                    tag "release-*"
                }
            }
        }
    }
}
```

```
        }

    }

steps {
    echo "Deploy to production."
}

}

}
```

ANY OF:

```
pipeline {

    agent any

    stages {
        stage("NewFeature") {

            when {
                anyOf {
                    branch "feature"
                    changelog "*new feature.*"
                }
            }

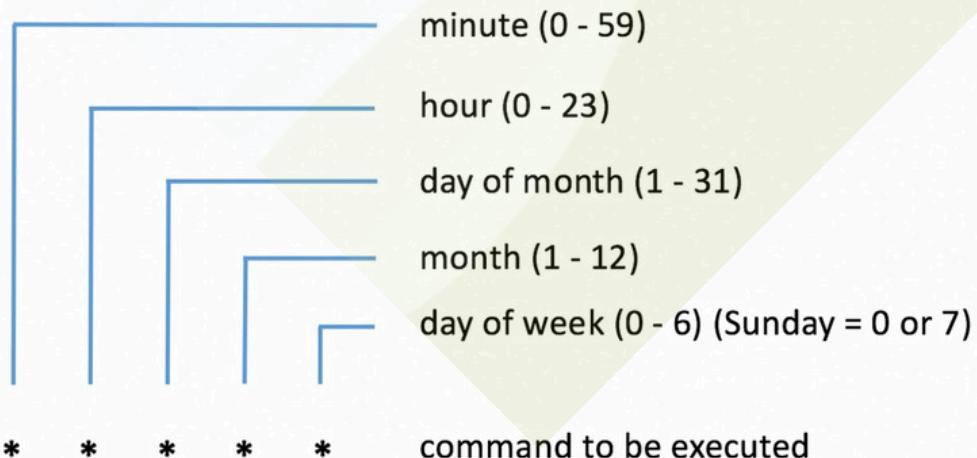
            steps {
                echo "Test new feature."
            }
        }
    }
}
```

BUILD PERIODICALLY:

```
pipeline {  
    agent any  
  
    triggers {  
        cron "* * * * *"  
    }  
  
    stages {  
        stage("Test") {  
            steps {  
                echo "Hello World!"  
            }  
        }  
    }  
}
```

CRON JOB IN FREE STYLE:

We can schedule the jobs that need to be run at a particular intervals.



Build Triggers --> Build periodically --> * * * * * --> save

If you want to make it customize then we can use cron syntax generator.

JENKINS PORT NUMBER CHANGE:

1. cd /usr/lib/systemd/system
2. vim jenkins.service
3. change port number in line 67
4. systemctl daemon-reload
5. systemctl restart jenkins