JENKINS

* It is an continuous integration tool. It is used to test and build software projects 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 you to continuously deliver software projects by providing powerful ways to define your build pipelines and integrating with large number of testing and deployment techniques.

**INSTALL JENKINS**

* To install Jenkins, you have download **Jenkins.war** file from the website.
* You can deploy the Jenkins.war file from cmd line (or) you can deploy it from the tomcat manager console.
* If you are deploying from console. You should have the .war file in the localhost.
* **Go to manager page.**
* **Mention context path (name).**
* **Mention War file path from the local machine.**
* **Click deploy.**
* It will be deployed automatically within few minutes.
* If you want to deploy it from cmd line, copy the war file to **webapps** dir.
* After deploying, click on the context path (name), it will take you to jenkins login page.
* It will ask you for password. By default, password will be stored in

**.jenkins/secrets/initialAdminPassword**.

* Copy the password and create an user jenkins jenkins. Now, we have succesfully installed jenkins.
* It will ask you to create an user to login to Jenkins. Crate a user, give password and mail.
* After creating user, you can login to jenkins with user.
* All the configurations and builds of jenkins are stored in **/root/.jenkins/workspace** dir.
* **/root/.jenkins/config.xml** – it will store all the configuration of jenkins.
* To change Jenkins **session timeout**.
* **Go to /root/.jenkins/war/war/WEB-INF/web.xml.**
* **Add time(min) in session timeout field.**

**CONFIGURE SYSTEM**

* **Home Directory** = Jenkins saves all the build files and settings in this directory. You can change this by exporting a new directory in your linux server where Jenkins is installed.
* **System Message** = this message will be shown at top of Jenkins dashboard.
* **# of Executors** = How many jobs can run in parallel in this server.
* **Usage** = How to use this server, as much as possible (or) only if builds are there matching this server label.
* **Quiet** **Period** = No of seconds to wait to start the job. It will put the job on hold upto that time.
* **SCM Checkout Retry Count =** No of times to check SCM, if it fails to checkout the code**.**
* **Environment** **Variables** = Set environment variables to tools.
* **Tool Locations** = Give tool locations to integrate those with Jenkins.
* **Timestamper** = Time format to show on console output.
* **Jenkins\_Url** = The Url to access Jenkins. It will come by default with localhost. If you want to access with domain name, you can give domain name in the field.
* **SystemAdmin e-mail** = You can give email address to send and receive mails for build actions.
* **Extended E-mail Notification** = Configure you mail to send mail to all reciepents when a specific build action occurs. You can select the mail format (text or html).
* **Email Notification** = Again Configure Email server settings and test the mail.

**NODES**

* We can configure slaves for jenkins master..means that we can build and test code in different environments (test-server), if it succeed we can deploy it in production. For this type of builds we configure slaves in jenkins.
* Jenkins slave/agents are just a java client process, which uses **JNLP** (java network launch protocol) to connect to master. Jnlp uses **49187** port. You can use random port if you don’t want to use this port.
* In **slave**, we have to install **java, maven** and **git** before building the projects. You can add as many slaves as you want.
* To configure slave.
* **Go to manage nodes.**
* **Click new node.**
* **Type a name to node.**
* **Select permanent agent** (or) **click copy from existing node** (if you already have a node).
* **Executors** = the no of builds thet jenkins may perform in this slave.
* **Description**, **remote dir**(where to save all the builds in slave).
* **Labels** = To identify slave.
* **Usage** = how to use this slave [as much as possible (or) only when build happens].
* **Launch** **method** = How to connect to slave (ssh, cmd from master).

If you select SSH, you have to give host ip, host user and password to open ssh connection from master to slave.

Before configuring slave, create an ssh connection between server and client to get rid of ssh errors while connecting for the first time,

* **Host key verify** = Controls how jenkins verifies ssh key presented by remote host.
* **Known host file verify** = It verifies known\_hosts file for the user, where jenkins is executed under (select this option if you are making an ssh connection with node).
* **Avaliability** = keep this host up all time (or) according to schedule (or) up when builds occur and die when no builds occurs.
* **Tool locations** = select java and build tools and specify home path for those tools. Click Save.
* Click , launch agent to start configuring the agent. Now, slave will be up and running in few minutes.
* Now, you can launch build projects in slave. It will create workspaces in remote directory where you specified while creating node.
* While creating project. Click on,
* **Where this project can be run** (to run this project on slave).
* **Type label of your slave.**
* **Select all other settings like git, build tools same as normal project.**
* **Click save.**
* Now, this build runs in slave and you can view it in remote directory once the build is completed.

**INTEGRATE-GIT**

* We use GIT in jenkins to pull code from git repository for build and test purpose. For this we have to integrate git with Jenkins and git should be installed in your Jenkins master server.
* To integrate Git,
* **Click manage Jenkins.**
* **Click manage plugins.**
* **Install GIT-plugin from avaliable list.**
* **Add git home dir in global tool configuration.**
* **Click, save.**
* To check, whether plugin is installed (or) not.
* **Go to projects.**
* **Create new free style project.**
* In the source management section. You will see **GIT** option, if you have installed and configured git plugin successfully.
* After integrated git with jenkins, you have to add your git repository while you creating a project. And you have to specify your branch.
* After adding your git repo, whenever you build a project, jenkins pulls the code from new commits from that repo and built it.
* There is an option in jenkins called **“POLL SCM”**, where you can set time(**CRON** **Expression**) for when jenkins should pull the code from git repository for the new commits to build.
* It will check the git repo for a specific period of time like we mentioned in poll scm, if there are any new commits it will pull that code and build the job.

**INTEGRATE SVN**

* You can integrate another scm tool **Subversion** with jenkins to automate the build jobs.
* You have to download the **subversion plugin** to integrate with jenkins. But, by default, subversion plugin will be downloaded while installing jenkins.
* So, you just have to configure subversion with jenkins by using **svn repo** and **credentials while creating job**.
* In the job, in scm section.
* Choose subversion.

**Repository Url** = specify your svn url along with your project path.

**Mention credentials** (which we are using to access svn server).

**Checkout strategy** = select the options based on your requirement (svn update, fresh copy etc).

* Now we are done with svn integration with jenkins.

**INTEGRATE-MAVEN**

* Maven is used to generate war and jar files. It means it will build the artifacts from the source code.
* We can use **MAVEN** to build in Jenkins (or) we can use **ANT**, which is another build tool.
* To integrate maven with Jenkins, first we have to install maven in server where Jenkins master resides and set home path for maven.
* To integrate Maven.
* **Click manage Jenkins.**
* **Click manage plugins**
* **Install maven-integration-plugin from avaliable list.**
* **Add maven tool and home dir in global tool configuration.**
* **Click, save.**
* To check, whether plugin is installed (or) not.
* **Go to projects.**
* **You will see a new project type called “MAVEN PROJECT”**
* If maven-project job is there you have succesfully installed maven. If not, the installation failed.
* While creating a maven-project, you have to specify the git repo and pom.xml file location. Only after this, it will start building.
* After every build jenkins creates a workspace in **/root/.jenkins/workspace** with project title and all the build files will be avaliable there.
* Whatever the process to build, we can see those steps in console output inside the project.

**JOBS**

* There are different types of jobs in Jenkins.
* **Freestyle project** = Used to create **ANT** based builds and normal deployment jobs.
* **Maven project** = used to create **MAVEN** based builds.
* **External job** = Used to build job in an remote machine, outside of Jenkins.
* **Multi-configuration job** = Used to run same build in different configurations and envs.
* All the build history of a project shown inside their respective jobs dashboard.
* To create a freestyle project.
* **Click new item.**
* **Type a project name.**
* **Select freestyle peoject.**
* **Mention all settings according to your build deployment.**
* **Click save.**
* Inside of a job dashboard,there are several options to discuss**.**
* There is an option called **discard old builds**. It means that it will rotate the old builds based on the specification we mentioned.
* If you enable this option, it will ask you

**Days to keep builds** = how many days of builds to keep (no).

**No of builds to keep** = How many builds to keep (no).

* You can disable a specific job by checking the **disable build** option.
* **Quiet period** = Jenkins starts build whenever a new commit occurs in SCM. But, if we set quiet period(sec), it will wait for specific time even a commit happened in SCM, only after quiet period is over, it will start the build. If a build happened in between the quite period, it will reset the quiet period time and it will start count down from the beginning.
* **Retry Count =** If we set Poll SCM, it will check the url for specific time for new commits and again it will check url only after the specified time mentioned in poll SCM. But, if we set retry count, it will check again immediately for specific no of times as we specified after regular checking is completed.
* **Custom Workspace** = Builds will save in default workspace, but if you want to save in different workspace, mentioned the workspace name here.
* **Trigger build remotely** = You can trigger the build remotely by using an url provided in Jenkins. You have to check this option before building remotely.

**JENKINS\_URL/job/job-name/build?token=TOKEN\_NAME**

* Token name is what we give in **AUTHENTICATION TOKEN** in master Jenkins.
* After building job remotely, in the console output it will show remote ip where the job is executed.
* **Build after other projects are build** = If we check this option and entered a job name, this job will run only after that specified job is completed like **downstream** **jobs**.

**Trigger only if build is stable =** This job will run if specified job is stable**.**

**Trigger even if build is unstable** =This job will run even if specified job will is unstable**.**

**Trigger even if build is failed** =This job will run even if specified job is fails.

* If you open the parent job, you can see child job as downstream job.
* If you open child job, you will see parent job as upstream job.
* If you run parent job, child job will run automatically after parent job is completed.
* Based on the settings you selected, child job will run after parent job resides in a specific state (stable, unstable, failed).

**GITHUB HOOK TRIGGER**

* We often push our changes to git repository. So, everytime we commited a change, we need to build the job manually (or) you can use poll SCM which will take time.
* For every commit we can’t build the job manually, we have to automate this process. To do this we need to install **github** plugin.
* After installing github plugin,

**Go to configure system.**

**In github section, click advanced.**

**Check “specify another hook url for github configuration”.**

**After checking the box, jenkins will give you an url along with web-hook.**

**Copy that url.**

**Go to your github account.**

**Select your reposiroty.**

**In repository, select settings.**

**Click, webhook option.**

**Click, Add webhook.**

**Paste the webhook url, which you have copied from jenkins.**

**Select the events to trigger this webhook (push, commit, delete etc).**

**Click save.**

* It will create an github webhook.
* Now, in the job configuration.

**In build triggers section,**

**Choose “GitHub hook trigger for GITScm polling” in new versions of jenkins.**

**Choose “Build when a change is pushed to github”………old versions.**

**Click save.**

* Now, when we push a commit, github sends a **post** **request** to jenkins that there is new code to build and jenkins will build the code according to the job.

**DASHBOARD**

* In the jobs dashboard, you will see all types of jobs created by all users.
* To separate and view jobs based on their type. We have to create another tab (or) view in jobs dashboard like java-build etc.
* Go to jobs dashboard.
* **Click + icon.**
* **Type tab name.**
* **Select view type (list view, my view).**

**List view** = shows items in simpler format.

**My view** = shows all jobs that current user have access to.

* After creating, go to that tab and click create new job, it automatically saves the job in this tab.
* To add existing job to this tab, click **add existing job** and select a job to add to this tab.
* To delete the tab, on the left pane, click **delete this view** to delete a tab.
* You can use **NESTED VIEW PLUGIN**, to create new view like folders and subfolders inside it, you can separate jobs based on the application.
* You have to download the nested view plugin and while creating views, select nested view and type a name to this new view.
* You can create subfolders in this view to separate the jobs. It will show separately inside the tab.

**BUILD-PIPELINE VIEW**

* In Jenkins, we can build jobs queue by triggering settings in parent and child jobs. It will build one after the other.
* With this pipeline plugin, you can see the process of running project one after the other. It will show you all projects in pipeline view.
* First you have to install **BUILD-PIPELINE PLUGIN**.
* **In jobs dashboard.**
* **Click + icon.**
* **Select pipeline view.**
* **Type a name to the view.**
* In the pipeline settings,
* **Select initial job** = from which job the pipeline will start. It will show you list of jobs present in your Jenkins master.
* **No of builds to display** = Select number.
* **Refresh frequency** = time in sec to refresh.
* After creating pipeline view, you can see the builds over pipeline view. Click run to start a job, you can see the all the jobs there running sequentially as we created.
* You can see your pipeline view in jobs dashboard by clicking **my views**. After clicking, you will see all your views (pipeline, list etc).

**USER ADMINISTRATION**

* In jenkins, you can create users and restrict users to specific jobs and specific permissions.
* There are several methods for creating users and groups in jenkins.
* **Jenkins-own-db** – You can create only users with this option, it saved in context.xml file.
* **LDAP** = You can use LDAP authentication for jenkins. You can login with your ldap username and password.
* **Unix user and password** = You can login with unix system users and groups where the jenkins master is installed.
* There are several types of permission sets for the users and groups.
* **Anyone can do anything** = means every one will have admin access, if you check this permission.
* **Logged-in-users can do anything** = logged-in-users have admin access for everything.
* **Matrix based security** = To give specific permissions to specific users.
* **Project-based matrix security =** To give specific permissions to specific users for specific projects. After enabling this security, you will get an option in the job dashboard called project based security.
* By this option, you can select which users should have what type of permissions for that specific job.
* **Role-based authorisation strategy** = With this option, you can create a role with specific permissions and you can attach this role users/groups for security concerns.
* All the users/groups who have got this role will get permissions what will role have and you can attach this role to a project. So, whatever the permissions the role have and whoever the users/groups attached to that role will get those permissions to that project.
* You have to install **“ROLE BASED AUTHORIZATION STRATEGY”** plugin.
* After installing that plugin, you will get a new option in manage jenkins **manage and assign roles.**
* You have to create a role with specific permissions and add users to that role to get those permissions.
* **Go to manage and assign roles,**
* **Click Manage roles,**
* **Create a role.**
* **Give specific permissions to that role.**
* **Click, Save.**
* Click Assign roles,
* **Add an user/group.**
* **Select a role to that user/group.**
* **Click, Save.**

**CONFIGURE SMTP IN JENKINS**

* We can configure smtp server in jenkins to get notified when a build action occurs.
* You have to download **Email-ext-plugin** and configure your email server in jenkins.
* Go to Manage jenkins,

**Click Configure system.**

* In extended email configuration,
* **Type your smtp server.**
* **Click advanced option.**
* **Select use smtp authentication.**
* **Type your mail, password, smtp port, ssl.**
* In email notification,
* **Type your smtp server.**
* **Click advanced option,**
* **Select use smtp authentication.**
* **Type your mail, password, smtp port, ssl.**
* Test the connection by clicking on test button and type your email id. If you get mail, you have configured smtp succesfully.
* While creating the project, at the bottom of the configure page, click **email notification** and type email ID’S to get notified whenever a build failure occurs.
* You can type any number of email in jenkins. It will send notifications to all mails.
* But, this option will send mails only when build failure occurs. It won’t send mails for successful builds.
* To send mail when build is success, there is another option.
* In post build action,
* **Click** **editable email notification**.
* **Project reciepient list** = type your emails separated with “,” .
* **Content type** = select how mail should look (html, plain text etc).
* **Default subject** = write down the subject as you like.
* Click advanced settings,
* **Click add trigger.**
* **Select trigger type (fail, success etc).**
* **Select reciepents list (to get emails)**
* **Click save.**
* You can add as many triggers as you want.
* Now, it will send mails regarding build success, fails, abort etc to all the reciepents as we mentioned.

**DEPLOY BUILDS TO CONTAINER**

* After the build is completed, the next big thing in the project would be deploy build files to conatainers (tomcat, jboss etc).
* Jenkins will deploy build files to container automatically, after building completes. We have to download **deploy-to-container-plugin** in jenkins.
* You have to create a new user and a new role in tomcat to deploy builds automatically to container through Jenkins and the role is **“manager-script”.**
* Go to **tomcatusers.xml** file**,** add a line in users section.

**<user username=”deploy” password=”deploy” roles=”manager-script”/>**

* Go to jenkins, go to your build job.
* In post build actions,

**Select Deploy war/ear to container.**

**Mention build file path = path/to/war (\*.war)**

**Add Credentials (which we created with manager-script role).**

**Add your container Path (url).**

**Add context path.**

**Click, save.**

* It will automatically build and deploy it to your container.
* Username and password for manager-script role can be anything. As for the test purpose I gave “**deploy**” for both username and password.
* Above is one method to deploy articrafts to container. But, generally In productions everyone uses separate jobs for both building and deploying purposes.
* For this type of deployments, You have to download the **CLONE WORKSPACE SCM PLUGIN.**
* This plugin will clone the build workspace dir to other jobs as we specified. By that cloned workspace, Jenkins will deploy the builds to containers.
* After installing clone plugin.

**Go to build(parent) job,**

* In post build actions,

**Select archieve for clone workspace SCM,**

**Files to include in workspace** = which files should be include in workspace(\*\*/\*.war).

**Criteria for build to be archieved** = most recent successful build (or) other options to deploy.

**Archieve method** =tar (or) zip**.**

**Click Save.**

* Now, create a **freestyle** job for deployment.
* In deployment job, you should select which build job should be cloned to deploy to container.
* Go to deployment job,
* In source code management section,

**Select clone workspace**.

**parent project =** which project to be deployed**.**

**Crieteria for parent build =** Select most recent successful build (or) other options**.**

* In post Build Actions,

**Select deploy to container.**

**Select all other settings of container to deploy (context path, url, war path).**

**Click save.**

* After configuring these settings, it will deploy the builds to container successfully. But, we have to run the build job and deployment job manually which will take lot of time to deploy. We can automate this process by enabling an option in parent (build) job.
* Go to build (parent) job,

**In post build action.**

**Select build other projects.**

**Select deployment project to build after this project automatically.**

**Select options to trigger build if parent job is stable (or) even it is unstable.**

**Click, Save.**

* After first job completion, in console output, you can see triggering new job line along with your child job.
* Whenever you build, Jenkins will deploy latest successful build to container as we configure in downstream job.

**BUILD METRICS**

* We can monitor all the builds in jenkins and about their states (success & failure).
* We have to download **BUILD-METRICS-PLUGIN**, and can see all the builds status with that plugin.
* After installing plugin.
* Go to manage Jenkins.
* you can see build metrics, specify the values like

**Show builds for the last** = select how days, months, years build to show.

* Select filtering,

**Node filtering, job filtering, job filtering, cause filtering** .

* By specifying these, you can see the status of all your builds and the states of your builds like how many build build failures occurred and how many successful builds etc.

**PARAMETERIZED REMOTE TRIGGER PLUGIN:**

* Jenkins can build jobs remotely without master-slave connection.
* There is a plugin called “**parameterized remote trigger plugin**” which is used to trigger builds remotely upon finishing job in local server.
* Download this plugin only in 1 server. Don’t need on remaining servers.
* **In the job configuration.**
* **Click build step.**
* **Select “Trigger a remote parameterized job” option.**
* **Override Credentials - choose credentials authentication. specify remote server credentials.**
* **Remote jobname or URl - Specify remote server job name (or) URl.**
* **Parameters - pass parameters if you want.**
* Save and build the job. The remote server job triggers automatically once server 1 job is finished.
* You can see the triggering in server1 job console output.

**DISK USAGE:**

* You can monitor disk usage by using jenkins. It will monitor the disk space which is being used by all the jobs.
* You have to download “**disk-usage**” plugin. Once you install the plugin you will see new option in jenkins dashboard.
* It will show you disk usage of builds, usage of jobs and usage of workspaces.
* By default, it will check your disk usage for every 6 hours. You can change this setting.
* Go to configure system.
* Scroll down to disk usage section, you will see cron expressions for monitoring disk usage for every 6 hours. Change this according top your requirement.
* The next option you will see “**show disk usage graph on project page**”. If you enable this option you can see disk usage graph in your project page.

AMAZON EC2 PLUGIN

* We can use amazon ec2 instances with Jenkins as slaves with **AMAZON EC2 PLUGIN**.
* It will work just as other local slaves in Jenkins. The advantage with ec2 is Jenkins will create ec2 instances when we start a job and terminates the instances after job completion (or) **idle time out** finishes, it will be an money saving idea for us.
* Jenkins will create instances from an AMI as we specified in Jenkins. So, whenever a new instance is started as slave, it is created by this specified AMI.
* By default, some instance types are not supported by Jenkins. So, Check whether the instance type is supported (or) not before creating cloud in jenkins.

**Prerequisites:**

**IAM User.**

**IAM User credentials (access and secret key).**

**Specific instance type .**

**AMI.**

**Security Group.**

**Remote user and remote dir (to store builds).**

**Region.**

**Set label, usage and idle timeout.**

* First create an IAM user and give him administrator access. Download the credentials of the IAM user.
* Launch an instance and install all the applications that you want to build (java is mandatory) and create an AMI from that instance.
* Go to Jenkins, Install **AMAZON EC2 PLUGIN**.

**Go to configure system,**

**At the end, click add new cloud**

**Give your aws iam user access and secret access keys.**

**Type your AMI ID.**

**Paste the pem file** and test the connection, the connection show success, if you have configured correctly.

**Type the security group name that you have used for AMI.**

**Select instance type** (not all types are supported with Jenkins)

**Select region and AZ.**

**Mention Jenkins home path where you want to save the builds.**

**Mention user to connect to ec2 from Jenkins** (with this user it will talk to ec2).

**Set label and set usage.**

**Set Idle timeout** (how much time instance should run. After this timeout completed, instance will be terminated even if there is a job running).

**Select no of executers.**

**Remote ssh port (22).**

**Instance cap** (how many instances can launch from an AMI. If you left this option blank, it will take as infinite).

* After selecting all these options.

**Go to manage nodes.**

**Select your slave.**

**Click configure.**

**Set tools home path as installed in your ec2 instance AMI.**

* while building a job, select this slave to run the job which is identified by a label and it will create an instance and build the job in it.
* After the idle timeout is completed, instance will terminate automatically.
* You can create multiple slaves with multiple AMI’s. Jenkins will bring up nodes according to your settings.
* You have to add another cloud, specify another AMI and mention new label and save.
* While creating job, select ec2 label. Then, it would run on the specific slave as we selected.

**CODE DEPLOY PLUGIN**

* With the help of this plugin, you can deploy your application from jenkins to aws ec2 instances.
* **Prerequisites:**

**Jenkins server**

**Ec2\_instance\_profile**

**Aws code deploy service role**

**Ec2 servers**

**Code deploy application**

**Deployment groups**

**Aws access and secret access keys**

**Region**

**Code deploy plugin (jenkins)**

**S3 Bucket**

* Create **ec2\_instance\_profile** role and code deploy **service** **role** in IAM service.
* Attach the ec2\_instance\_profile role to ec2 servers while creating them.
* After creating ec2 servers, login to those servers and install code deploy agent in all the servers.
* Create an code deploy application and attach code deploy service role while creating application.
* create an deployment group inside code deploy app with multiple ec2 servers as you want.
* Create an **S3** bucket, where code deploy will store all your code in zip format.
* Download your **access** **key** and **secret** **access** **key**.
* Write **appspec.yml**, **start** and **stop** scripts, on how and where you want to deploy your code. These files should be present along with code in SCM.
* Now go to jenkins, install **code**-**deploy** **plugin**.
* After installing plugin, you will get code deploy option in **configure** **system**.
* Specify your aws keys in configure system in their respective fields.
* Now in the job, in post-build action, choose **“Deploy an application to code deploy”.**
* Specify all the settings as the jenkins prefered.
* **CodeDeploy Application Name** = name of your CD app in aws
* **CodeDeploy Deployment Group** = name of your deployment group in aws CD
* **CodeDeploy Deployment config** = specify how deployment should happen
* **Aws Region** = region where your ec2 servers and CD are present.
* **S3 Bucket** = specify bucket name to store your code.
* **Aws Access key** = specify your aws account access key
* **Aws Secret Access key** = specify your aws account secret access key
* Save the job and build the job.
* Once we build the job, code deploy will zip the code and deploy the zip file to S3 and deploy it to all the ec2 servers by unzipping it based on our appspec.yml file.
* If you enable versioning in S3, it will store your zip files as versions.
* If the deployment fails, code deploy automatically rollbacks to the old application revision.
* You can see the deployment logs in code deploy console. It will provide you the logs for troubleshooting.
* For each and every deployment, aws provides an ID for identification. You code will be downloaded into your ec2 servers in **/opt/** dir. You can check your code by deployment ID