***JENKINS***

Open source automation tool is written in JAVA with plugins built for Continuous Integration & Deployment purpose.

Stages of CI/CD in Jenkins Job: **PULL > BUILD > DEPLOY > SANITY TEST**

***CI - Continuous Integration*** - Integrate the changes from the development team continuously without manual intervention. Each integration must be verified by automation build and integration tests and provides quick feedback to the developers.

***CD -Continuous Delivery*** - Deploys the build to the test environments and performs a series of tests. If no issues are reported, wait for manager/CAB (Change Advisory Board) approval for production deployment.

* Provides capability to deploy build to any environment at any given time.

***CD - Continuous Deployment*** - Deploys the build to the test environments and performs a series of tests. If no issues are reported, then directly deploy the code to production without any approvals and manual interventions.

***Differences – CI-CD CD***

|  |  |  |
| --- | --- | --- |
| ***Continuous Integration*** | ***Continuous Delivery*** | ***Continuous Deploy*** |
| Integrating changes manually without manual intervention | Automatically deploy the code changes on different test environments. Ready to deploy it on Prod environment at any time. | Every small change will be tested in different test environments and in case no issues are found automatically deploys it on Prod |
| Immediately tested and feedback is sent to developers | Changes are executed in environments and stages – like unit, integration, QA, dev, and pre-prods’ UAT. | |
| Starts every time the code is pushed to the repo and performs the unit and integration tests | If UAT is successful, waits for managers’ approval to deploy the code on the PROD | On successful UAT, the code is directly deployed to the environment and runs the sanity tests. |
|  | The code will be deployed on PROD with a scheduled downtime. Practiced with the applications which can have downtime(noncritical applications with less traffic) | Used in the case of applications that demand zero downtime. |

***Jenkins Process***

* Once the code changes are pushed to Git hub/ SCM tool, the Jenkins pulls the changes immediately and triggers the Jenkins job ::: build -> compile -> deploy -> deliver.
* Jenkins can integrate all the processes and tools and run any script in the scheduled time.

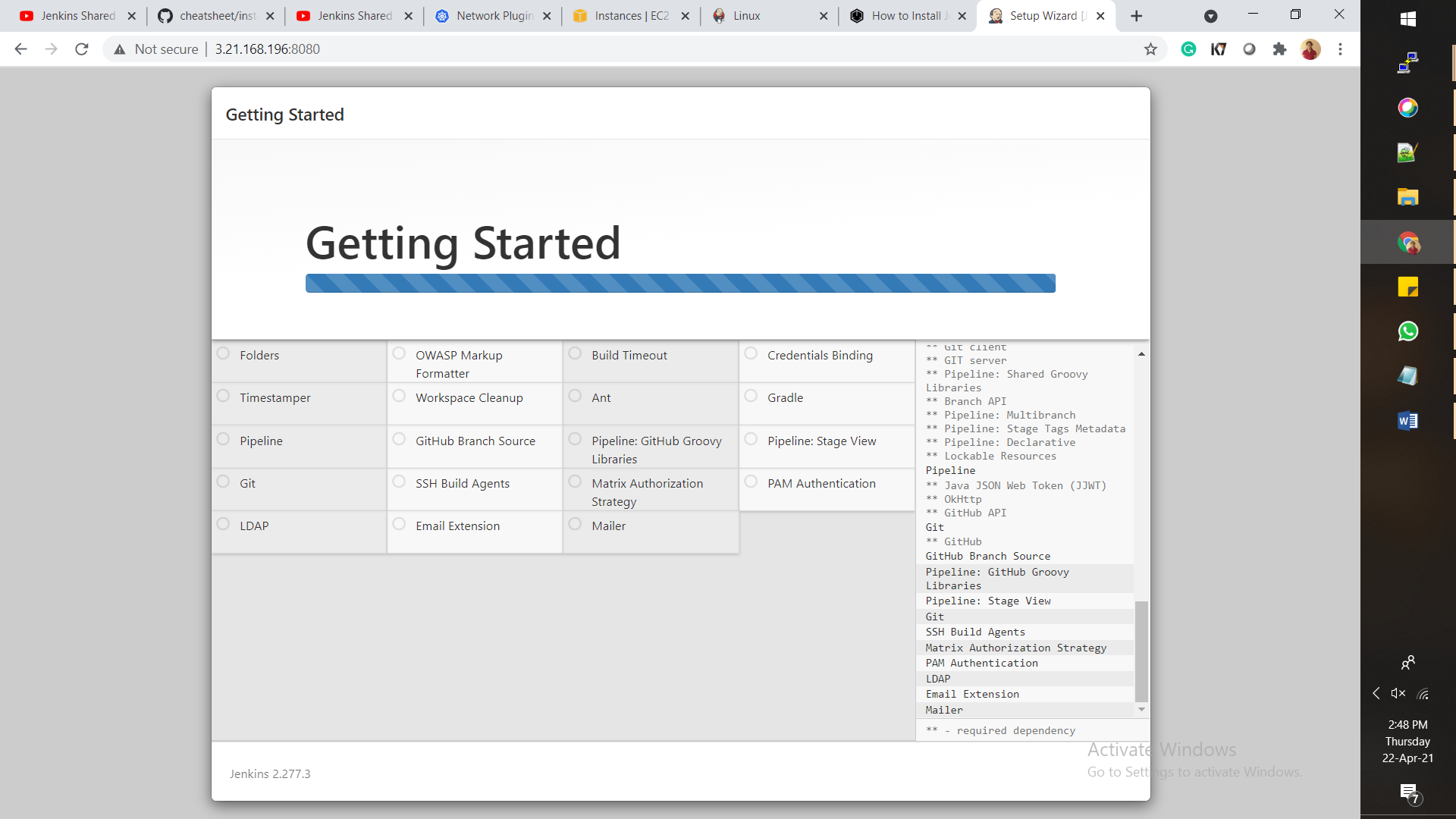
***Jenkins Installation***

1. Install JAVA
   1. sudo apt install openjdk-11-jdk
2. Install Jenkins
   1. wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
   2. Sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \

/etc/apt/sources.list.d/jenkins.list'

* 1. Sudo apt-get update
  2. Sudo apt-get install Jenkins

1. Start Jenkins - ***systemctl start Jenkins –*** *Jenkins takes* ***8080*** *port by default*
2. ***cat /var/lib/Jenkins/secrets/initial admin password*** – for initial password
3. Install default plugins



**Manage Jenkins** – Give tools for system configurations and security

**Configure System** – configure settings for execution details related to all other integrated apps.

**Global Tool configuration** – sets paths in instance and credentials for integrated tools

**Manage Plugins** – Gives all plugin details Installed, Available, etc.

**Manage Nodes and Clouds** – Adds removes and monitors Slave nodes.

**Manage Credentials** – Manages credentials for Jenkins and other integrated systems

**Manage Users** – Manages different users

***Authorization in JENKINS***

*1. Anyone can do anything*

Everyone gets full control of Jenkins, including anonymous users who haven’t logged in. Do not use this setting for anything other than local test Jenkins controllers.

*2. Legacy mode*

Behaves the same as Jenkins <1.164. Namely, if a user has the "admin" role, they will be granted full control over the system, and otherwise (including anonymous users) will only have read access. Do not use this setting for anything other than local test Jenkins controllers.

*3. Logged in users can do anything*

In this mode, every logged-in user gets full control of Jenkins. Depending on an advanced option, anonymous users get read access to Jenkins or no access at all. This mode is useful to force users to log in before taking actions so that there is an audit trail of users' actions.

*4. Matrix-based security*

This authorization scheme allows for granular control over which users and groups can perform which actions in the Jenkins environment (see the screenshot below).

User/group | Overall, Credentials, Agent, Job, Run, View, SCM, Lockable Resources

Administer,Read,Create,Delete,ManageDomains,Update,View,Build,Workspace,Configure,Connect,Disconnect,Tag,Reserve

*5. Project-based Matrix Authorization Strategy*

This authorization scheme is an extension to Matrix-based security which allows additional access control lists (ACLs) to be defined for each project separately in the Project configuration screen. This allows granting specific users or groups access only to specified projects, instead of all projects in the Jenkins environment. The ACLs defined with Project-based Matrix Authorization are additive such that access grants defined in the Configure Global Security screen will be combined with project-specific ACLs.

*6. Role-based Authorization Strategy – PLUGIN*

Helps in Security Authentication and User Management

Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.

***Parameterized Build***

General - > This project is parameterized – we can give key-value pairs for these parameters & build with parameters

1. Boolean parameter ->

2. Choice parameter ->

3. Credentials Parameter -> Lets you give/choose Agents and tir credentials during runtime

4. Fie parameter

5. Git parameter

6. JFrog pipelines

7. Jeera issues

8. Jeera version

9. Label

10. Multi-Line string nodes

11. Node

12. Non stored password

13. Password

14. Run

15. String

***Types of Jobs***

1. Freestyle projects
2. Pipeline
3. Maven
4. Multi configuration
5. Multibranch Pipeline

***Freestyle – JOB page***

***General*** – we can set some prerequisites here – like discard old build, parameters, execute concurrent builds.

***SCM*** – Non / GIT – git repo URL & creds, build branches. , additional like setting tags, merging before build cleanup unwanted tags, clone.

***Build Triggers***

1. Trigger builds remotely
2. Build after other projects are built
3. Build Periodically – like CRON – but builds in every time the job run
4. Poll SCM – Run the job acc to CRON set but builds only if new commits are identified
5. GitHub hook trigger for GITScm polling – A webhook will be set in Git Hub and every time a new commit is made the GIT sends a notification to Jenkins and Jenkins to pull the new commit and builds.

***Build Environment***

* Sets environment features expected before the build is triggered.

***Build –*** Gives the build instructions

* Execute shell, invoke windows Batch command, top-level maven objects.

***Post Build*** – instructions to be followed post-build

* To run dependent build after build completes, cleanup workspace, email notification, etc.

***PIPELINE jobs***

We have to specify the job configuration in the groovy script.

The script can be written in the Pipeline definition or can be pulled from the GIT hub.

***Maven project***

Build triggers – Same as freestyle with few additions like – Dependency update trigger, snapshot builds

Also, the host machine should have maven installation, and the path and version need to be specified in Jenkins global tools configurations.

*Root POM* – pom.xml

*Goals and options* should be set with clear Maven instructions

***Multi-branch Jenkins job***

The Multibranch Pipeline project type enables you to implement different Jenkinsfiles for different branches of the same project. In a Multibranch Pipeline project, Jenkins automatically discovers, manages, and executes Pipelines for branches that contain a Jenkinsfile in source control.

***Master and Slave setup***

Pre-requisite – ***1 Master*** node hosting the Jenkins environment. ***At least 1 Agent*** node with the run dependencies installed like GIT, maven, GCC, java, etc.

*Go to Manage Jenkins -> Manage Nodes*

1. Default Master will be present.
2. New node - > Agent-1 Permanent Node
3. No. of executors – this no. specifies no. of scripts that can be executed parallel in the agent.
4. Remote directory – Specific dir on agent nodes dedicated for Jenkins execution ***/var/Jenkins***
5. Labels – identifies of the agent in master job
6. Usage – i. As much as possible ii. Only when labels are matching
7. Launch Method – SSH

Host id – Agent IP

Username

Private Key – Enter key directly

1. For thekey -> Goto Agent machine -> run the cmd

***ssh-keygen -t rsa -C "The access key for Jenkins slaves"***

It will be saved in location - ***/home/ubuntu/.ssh/id\_rsa***

***cat id\_rsa***

1. Take the complete ID generated and enter in key.
2. Trust method – manual
3. Save and close
4. Launch Agent – Trust verification manually and launch again
5. Connection should be useful
6. During Build – Mention the ***label*** of the slave in Prerequisite

# *Jenkins Automated Build Trigger on Github Pull Request*

1. *Manage Jenkins -> Manage plugins -> GitHub Pull Request Builder*
2. Configure System -> *GitHub Pull Request Builder*
3. *GitHub URL for the project to be built*
4. Kind – Username & Password – Provide GitHub credentials
5. Connect to API
6. Save and apply
7. Goto GitHub -> Settings webhooks -> payload URL
8. Events to trigger the webhook - Show individual options-> Pull request

***Jenkins Shared Library***

Jenkins shared library is the concept of having common pipeline code in the version control system that can be used by any number of pipelines just by referring to it.

Eg – if we have 10 java microservices pipelines the maven build step will be duplicated in all 10 pipelines. In addition to new services, we have to copy-paste pipeline code again. If we want to change some parameters, we will need to manually do it in all the pipelines.

When we use the shared library all the common code is kept there and if we update the code there it will appear in all the 10 pipelines.

Jenkins shared library

1. vars – This directory holds all the global shared library code that can be called from a pipeline. It has all the library files with .groovy extn.

2. src – It’s a regular java source directory. It is added to the classpath during every script compilation. Here we can add the custom groovy code to extend the shared library code.

3. Resources – all the non-groovy files required for the pipeline can be managed from here. Like Jenkins file, XML, etc.

