**DevOps**

**Waterfall model :**

It has distinct goals for each phase of development. Imagine a waterfall on the cliff of a steep mountain. Once the has flowed over the edge of the cliff, it can not turn back.

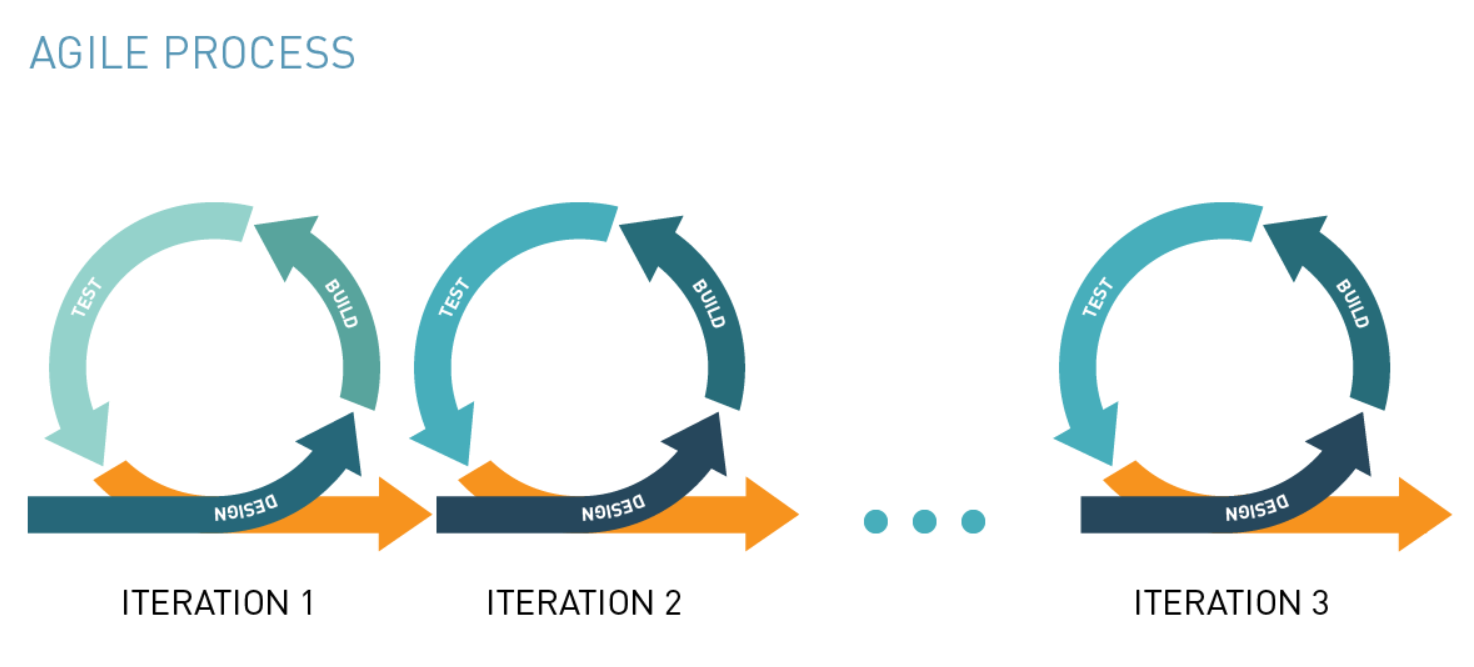
Requirement gathering & analysis 🡺 Design 🡺Implement 🡺Test 🡺 Deploy 🡺 Maintenance

**Disadvantages of waterfall model:**

* Once the application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* Not a good model for complex and object oriented projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.
* High amount of risk and uncertainty.

**Agile methodology :**

* In the agile methodology each project is broken up into several iterations.
* All iterations should be of the same time duration (between 2 to 8 weeks)
* At the end of each iteration, a working product should be delivered.
* So agile methodology breaks down software delivery life cycle in to several iterations or sprint involving continues delivery and testing of the software delivery .



**Limitations of Agile 🡺**

* Codes works in developer’s system but not in production.

**Devops :**

Devops is a practice of operations and development engineers.

**Devops stages:**

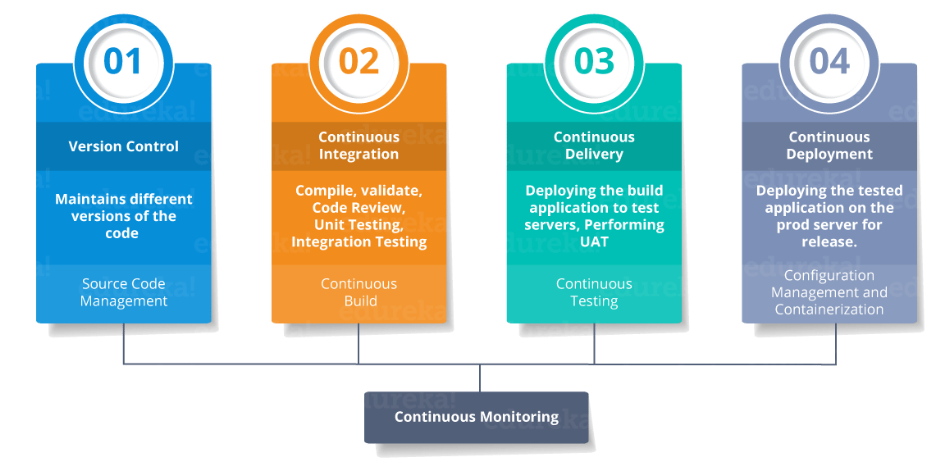
**Version control🡺continues integration 🡺 continues delivery 🡺 continues deployment**

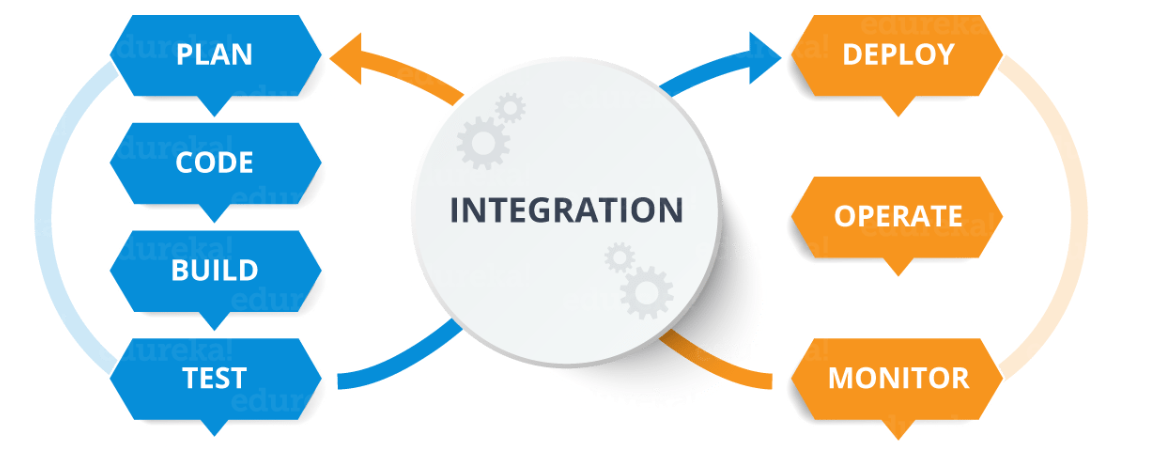
**Version control** : maintains different versions of the code.

**Continues integration** : compile, validate, code review, unit testing, integration testing, packaging

**Continues delivery**: deploying the build application to test servers, performing UAT

**Continues deployment**: deploying the tested application on the prod server for release.





**Plan & code :** git, subversion, jira

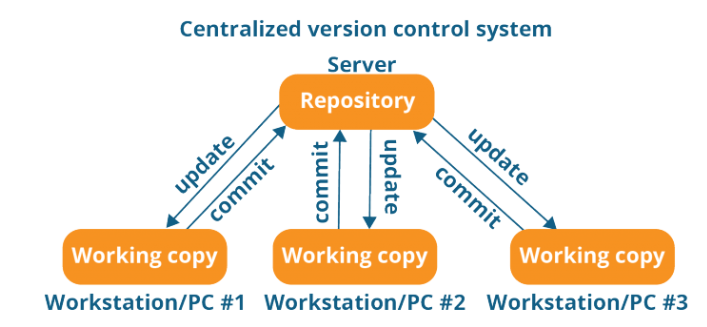
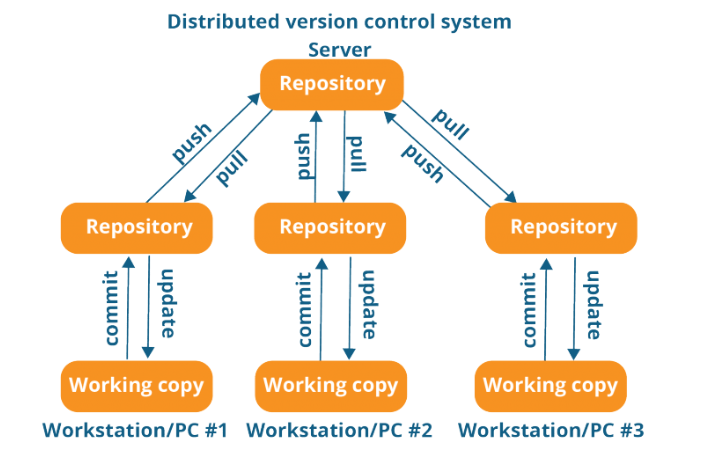
**Build** : maven

**Test** : Junit, gradle

**Integration** : Jenkins

**Source code management :**

Version control system is of two types.

* Centralized version control system
* Distributed version control system

Creating repository 🡺

mkdir devops

git init 🡺 initialize git repository

git status 🡺 status of the git repository.

git add a.txt b.txt 🡺 add two files in staging area

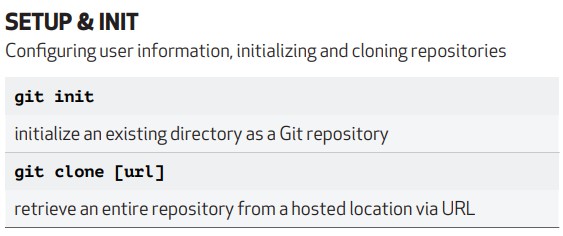
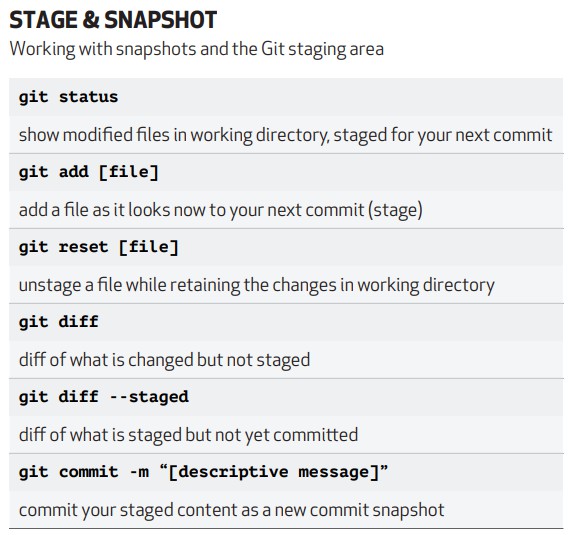
git . 🡺 add all files

git commit -m “first commit comment here” 🡺 commit the latest changes to main branch

SETUP

Table

Description automatically generated

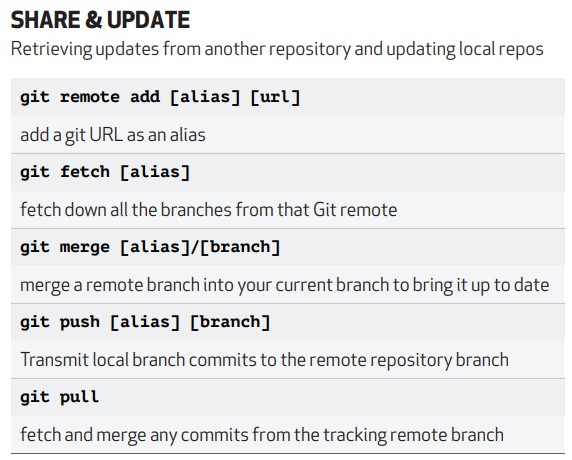


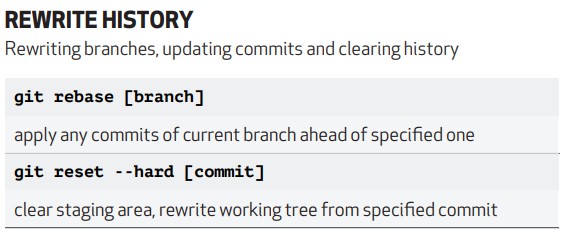
Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated





Graphical user interface, text, application

Description automatically generated

**JENKINS**

**What is Jenkins :**

* Jenkins is the most popular open source continuous integration and continuous delivery server capable of orchestrating a chain of events/actions that help to achieve CI in an automated fashion.
* What is continuous integration : Continuous integration is nothing but integrating all devops cycle tools
* Jenkins is a free open source Continuous Integration tool and automation server to monitor continuous integration and delivery. It is written in Java.
* It is known as an automated Continuous Delivery tool that helps to build and test the software system with easy integration of changes to the system. Jenkins follows Groovy Scripting.
* Also, it enables developers to continuously check in their code and also analyze the post-build actions. The automation testers can use to run their tests as soon as the new code is added or code is modified.
* This CI server runs in servlet containers such as Apache Tomcat

**Advantages of Jenkins :**

* Automated build and testing
* Instant feedback to developer
* Low risk and faster delivery
* frequent commits and small feature release

**Continuous integration :**

CI is a development practice where developers integrate code into a shared repo frequently. Each integration can then be verified by an automated build and automated test.

* One of the key benefits of integrating regularly is that you can detect error quickly and locate them more easily.

**Continuous delivery VS Continuous Deployment**

* In case of continuous deployment the process of deployment to production is automated after acceptance testing.

code checking 🡺 unit testing 🡺 code QC 🡺 Build and deployment 🡺 Acceptance testing 🡺 Prod deployment

**JENKINS INSTALLATION** 🡺

* install java and setup system variable
* download Jenkins installable and install Jenkins
* Authenticate first time use
* Setup and install Jenkins plugin

**Architecture of Jenkins**

Before we dive into how Jenkins works, we must understand the architecture of Jenkins. These are the series of steps that outlines the interaction between different elements in Jenkins:

* **Developers do the necessary modifications in the source code** and commit the changes to the repository. A new version of that file will be created in the version control system that is used for maintaining the repository of source code.
* **The repository is continuously checked by the Jenkins CI server for any changes** (either in the form of code or libraries) and changes are pulled by the server.
* **In the next step, we ensure that the build with the ‘pulled changes’ is going through or not**. The Build server performs a build with the code and an executable is generated if the build process is successful. In case of a build failure, an automated email with a link to build logs and other build artifacts is sent to the developer.
* **In case of a successful build, the built application (or executable) is deployed to the test server**. This step helps in realizing continuous testing where the newly built executable goes through a series of automated tests. Developers are alerted in case the changes have caused any breakage in functionality.
* If **there are no build, integration, and testing issues** with the checked-in code, the changes and tested application are **automatically deployed to the Prod/Production server**.

Here is the diagrammatic representation of the Jenkins architecture:



* A single Jenkins server might not be sufficient to realize the following requirements:
* Testing needs to be performed on different environments (i.e. code written using different languages e.g. Java, Python, C, etc. are committed to the version control system), where a single server might not suffice the requirement.
* A single Jenkins server might not be sufficient to handle the load that comes with large-scale software projects.
* In such scenarios, the distributed (or Master-Agent) architecture of Jenkins is used for continuous integration and testing. Diving deeper into how does Jenkins works, we take a look at the architecture of Jenkins.

**Master- Agent Architecture In Jenkins**

* The master-agent (or distributed) architecture in Jenkins is used for managing distributed builds.
* The Master and Agent(s) communicate through the TCP/IP protocol.

These are the roles and responsibilities of the Jenkins Master and Agent(s):

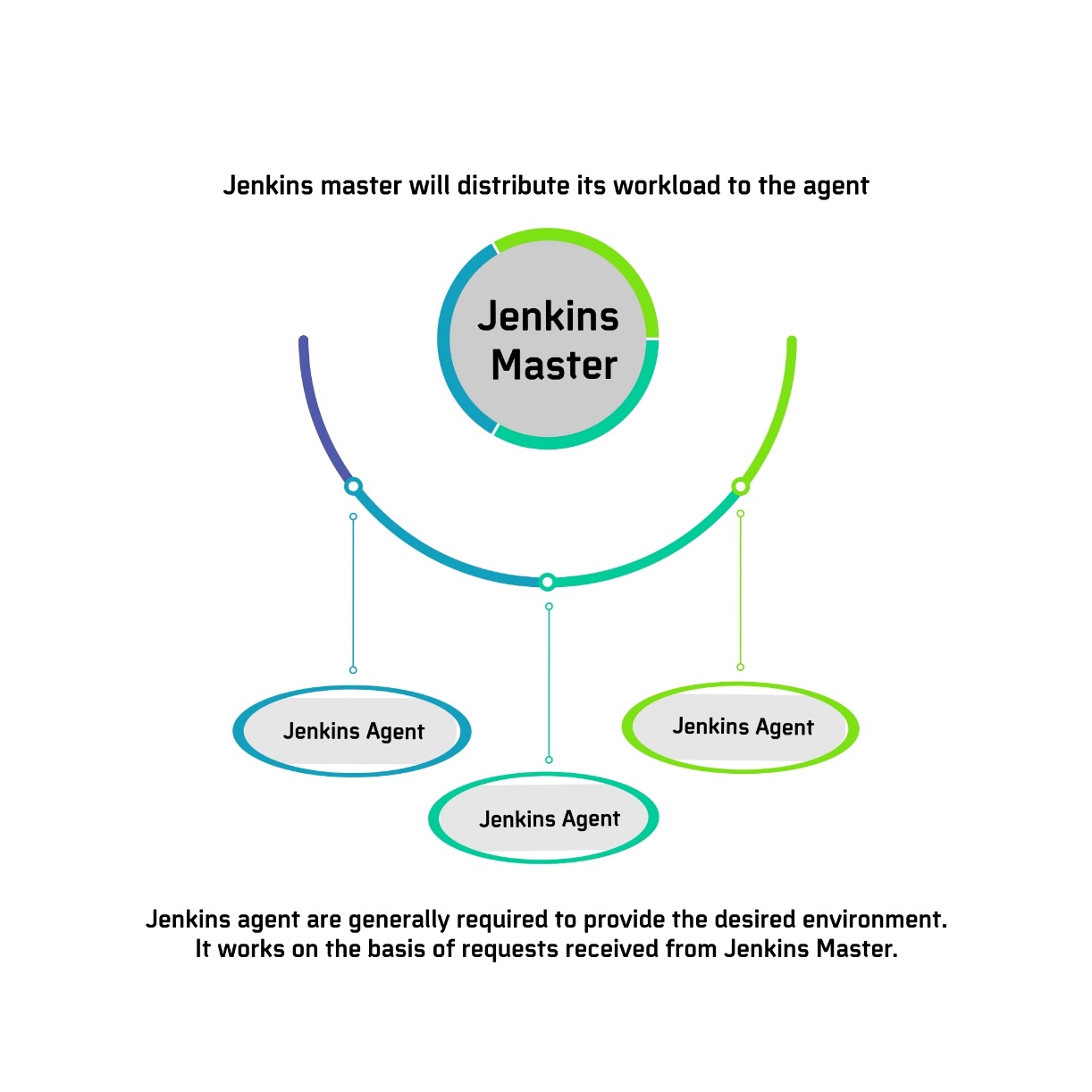
**Jenkins Master**

* **The main server in Jenkins is the Master**. Here are the jobs handled by Jenkins Master:
* **Schedule build jobs**
* **Choosing the appropriate agent in the master-agent ecosystem** for dispatching the builds.
* **Monitor agents** and take them online/offline as and when required.
* **Presenting the build results** (and reports) to the developer.
* The Jenkins master can also execute the jobs directly, but it is always recommended to select the appropriate agent(s) for build and execution-related tasks.

**Jenkins Agent(s)**

* A agent is a remote machine that is connected to the Master. Depending on the project and build requirements, you could opt for ‘N’ number of agents.
* agents can run on different operating systems and depending on the ‘type of build request’, the appropriate Agent is chosen by the Master for build execution and testing.
* Here are the jobs handled by the Jenkins Agent(s):
  + - Listen to commands from the Jenkins Master.
    - Execute build jobs that are dispatched by the Master.
    - Developers have the flexibility to run the build and execute tests on a particular agent or a particular type of Agent. The default option is Jenkins Master selecting the best-suited Agent for the job.

Here is a simple diagrammatic representation of how does Jenkins work, with multiple Jenkins Agents connected to the Jenkins Master:



**How Does Jenkins Work In Master-Agent Architecture?**

* In the Jenkins Master-Agent architecture shown below, there are three Agents, each running on a different operating system (i.e. Windows 10, Linux, and Mac OS).
* Developers check-in their respective code changes in ‘The Remote Source Code Repository’ that is depicted on the left-hand side.
* Only the Jenkins master is connected to the repository, and it checks for code-changes (in the repository) at periodic intervals. All the Jenkins Agents are connected to the Jenkins Master.
* Jenkins master dispatches the request (for build and test) to the appropriate Jenkins Agent depending on the environment required for performing the build. This lets you perform builds and execute tests in different environments across the entire architecture.
* The Agent performs the testing, generates test reports, and sends the same to the Jenkins Master for monitoring.
* As developers keep pushing code, Jenkins Agents can run different builds versions of the code for different platforms.
* jenkins Master (or Master Node) controls how the respective builds should operate.

Diagram, application

Description automatically generated

**Jenkins Dash board:**

* **Job/Project** : It refers to the tasks that are configured in jenkis.
* **Nodes** : Each server/machine that is part of Jenkins
* **Executer** : Thread or slot for execution of jobs.
* **Build** : result got after executing a job
* **Plugin** : software that extends core functionality of Jenkins.

**Left side tool :**

* new item
* people
* build history
* manage Jenkins > configuration system > configure global security is used to manage user
* my views
* credentials
* new view

manage Jenkins > configutaion system > configure global security is used to manage user

manage plugin : install /uninstall plugin

My views : personalized view of any iser . He can setup tabs as per his requirement.

Build executer status : Shows current jobs those are being executed on diffret executer.

Build queue : Build jobs those are triggered but waiting for queue to get started.

**Status** :

* **Blue** : Success
* **Grey** : Not built
* **Red** : failed

**Weather** :

* **Sun : good**
* **Cloud : failed few times**
* **thunderstorm : failed multiple times**

**Role based strategy / creating users & Granting access :**

* Jenkins > manage plugins > search role-based authentication strategy
* manage Jenkins> configure global security > check on role based strategy
* go to manage and assign roles
* you can create roles here.
* Note: users can be given to role bases strategy only after the users are created

**Managing Nodes on Jenkins 🡺**

Nodes are nothing but different server, on which jobs will be executed or integration will happen

* Go to manage Jenkins > configure global security > Agents
* Click on Random
* Go to Manage Nodes > New node
* Remote root directory > mention the directory name where you want to keep all your repo
* launch method: jave webstart > save

agents.jar and slave-agent.jnlp

* login to the server that you want to make a node.
* put the agents.jar and slave-agent.jnlp file in the server in the location you have mentioned in the Jenkins setup process for this node.
* go to slave-1 or node-1 or node that you have created in Jenkins, you would get one command shown to be executed in agent slave. Copy that and execute in created note server.

**How to create user:**

create user > username >pwd>full name > email

* configure global security
* check on enable security
* security realm > is the process to authenticate pwd, this is to enable the user to login through password.

**Matrix based security**: this allows to give access to the user with matrix like read/delete/create for any component like job/agent/run

Access to any user can be given in matrix-based security. This enters the user permission in Jenkins.

**Creating Jenkins jobs** 🡺

* New item> free style project
* Discard old build : how long you would like to keep old build history
* Git hub project :
* This project is parameterized : This indicates which exact instance you would like to run
* **Throttle build** : It defines how many times the job will run in a particular time.
* No of throttle 1 and time period 60 🡺 once in 60 min
* no of throttle 2 and time period 60 🡺 twice in 60 mins in 30 mins of interval

Execute concurrent builds if necessary 🡺 Enabling this will execute multiple build of same job in different instance.

**Build triggers :**

Build periodization :

min Hour Date Month Day\_of\_the week

\* \* \* \* \*.

0-59 0-23 0-31 0-12 0-7

1. Sunday
2. Monday
3. Tuesday
4. Wednesday
5. Thursday
6. Friday
7. Saturday
8. Sunday

@hourly 🡺 hour;y

H/15 🡺 every 15 mins

Build : triggers builds remotely , it will help to call a job from outside

**How to execute a job from another job :**

Job 1 JOB2 Job3

* go to job config
* go to build trigger
* **check on build after other projects are built> master job name**

**GIT Integration with Jenkins**

1. **Poll SCM**
2. **Github webhook**

* **Poll scm : Jenkins will keep polling GIT to check if there are any new check-ins**
* **Git hub webhook : Jenkins will listen to a webhook and git will let Jenkins know if there are any new check-ins**

**POLL SCM :**

* Install git plugin > manage Jenkins>manage plugin > git plugin
* Copy the repo link in git hub
* go to job in Jenkins > right clock > configure > check in git hub project, paste the repo url (browser address)
* go to source code management > git > paste repo link
* check in poll scm in build trigger > schedule it > it should be

min hour date month day-of-week

**Git webhook :**

perquisites: You need to have a public IP for Jenkins link. Local host will not work. You can use ngrock to create public link of your Jenkins local address.

EX: go to cmd > type the command > ngrock http 8080

* Go to Git hub > setting > webhook
* add webhook > apply link > paste the Jenkins URL and add github\_webhook in the last
* EX : If the Jenkins link is <http://xyz.com> you need to add <http://xyz.com/git_webhook>
* Go to Jenkins > job>configure>check in git hub webhook trigger for git scm polling
* do not select poll scm
* source code management > paste the repo link

**Difference between POLL SCM & WEBHOOK ?**

**Poll SCM 🡺**

* Resource intensive
* Heavy operation and needs to be fired frequently
* Wait for builds. Have to wait in the order of minutes for the build to happen as you have schedule it and it will execute only during that scheduled time
* Safer option. Safer as Jenkins is communicating to code repo.

**Webhook 🡺**

* Not costly. No constant checking involved
* instantaneous build. Builds are triggered immediately whenever a check in happen
* Security concerns as opening Jenkins to the outside world.

**Maven Integration in Jenkins 🡺**

Maven is a build tool, which is used to package your file to jar or war

* Download maven and install
* Setup env variable > right click on my computer > system properties > advanced > env variables > new
* M2\_HOME 🡺 values 🡺 path of the maven folder that downloaded
* MAVEN\_HOME 🡺 values 🡺 path of the maven installation folder like programe files path
* update PATH variable > click on PATH > edit > copy the bin folder address of the maven folder (downloaded> and add it in the last of variable value

**There two ways, maven can be integrated into Jenkins**

* Using maven integration plugin : Job configuration > build environment > check in delete work space before build starts
* build > involve top level maven targets > goal >description >POM > location of pom.xml file

Another way is to integrate through plugin

* manage plug in > maven integration plugin > new item > maven project > build environment > delete workspace before build starts
* build > root POM > pom.xml
* goals and options : clean test package
* Advanced > Enable triggering of the down stream project.
* check in use custom workspace
* directory : copy the path of the workspace directory
* check in : resolve dependencies during POM packaging
* Jenkins configuration > global tool configuration > verify jdk location and all the details

**Sending EMAIL notification from Jenkins** 🡺

* Jenkins > configuration > email notification
* smtp server : smtp.gmail.com
* default user email : @gmail.com
* usernme : [abc@gmail.com](mailto:abc@gmail.com)
* password : abc@123
* user SSL : checkin
* smtp port : 465
* reply to address : [abc@gmail.com](mailto:abc@gmail.com)
* carset : utf-8
* Test configuration

We have to change the system admin email address in Jenkins location . That means the mail will be from the mentioned email id. system admin email id

\*from job we can set email notification > go to job > post build actions > email notifications

There are few drawback of email notifications. To advance all this, we hve another email plugin.

manage Jenkins> plugin > email extension plugin >

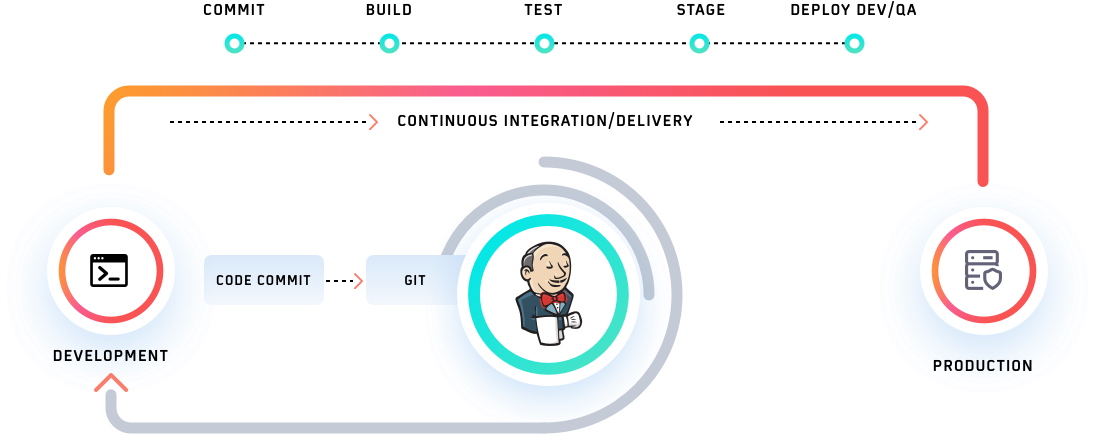
configure syetm > extentded email nitification

smtp server > default email suffix >user>pwd

default trigger > click on which ever is required

Advance settings > uncheck developer from trigger setting

**Pipeline Jobs in Jenkins :**



**There are two types of pipeline jobs in Jenkins.**

* **Delivery pipeline**
* **Jenkins pipeline**

**Delivery pipeline :**

* **you can create multiple freestyle project or any job.**
* **once you create jobs, go to configuration of jobs (each job).**
* **go to build trigger section and check build after other jobs are built.**
* **install delivery pipeline plugin to visualize the pipeline jobs**
* **got to view, create a view > delivery pipeline view> enter view name > got to pipeline component mention the initial job .**

**ex: we have created 3 jobs. job1 job2 job3**

**we can go to job3 and set trigger as build after other jobs are build and mention job2.**

**like wise we can setup for job 2.**

**Jenkins Pipeline 🡺**

**Jenkins pipeline is a combination of plugins that supports integration and implementation of continuous delivery pipelines. A Pipeline is a group of events interlinked with each other in a sequence.**

**The Jenkins pipeline has an expandable automation system for building basic or complicated ‘template’ distribution pipelines via the Domain-specific language (DSL) used in the pipeline. There are four states of Continuous Delivery in Jenkins pipeline-**

* **Build**
* **Deploy**
* **Test**
* **Release**

**We are now aware that Jenkins has proved to be a specialist in Continuous Integration, Continuous Testing, and Continuous Delivery. It uses a feature called Jenkins pipeline for Continuous Delivery, which is basically the ability to release apps regularly at an interval. This process ensures that the software is always ready for production**.

**Jenkins File :**

* **Jenkins pipelines can be defined using a test file called JenkinsFile.**
* **Jenkinsfile is just a text file, usually checked in along with the project’s source code in Git repo.**
* **Ideally, every application will have its own Jenkinsfile**
* **You can implement pipeline as code using JenkinsFIle and this can be defined by using a domain specific language (DSL).**
* **in Jenkinsfile you can write the steps needed for running a jenkins pipeline.**
* **JenkinsFIle can be defined by either webUI or with a Jenkins File.**

There are two types of Jenkins pipeline syntax for defining your JenkinsFile.

* Declarative
* Scripted

**Pipeline 🡺** The Pipeline consists of a set of instructions written as code. It defines the entire build process, which ideally consists of different stages for building, testing, and delivering the application.

**Node 🡺** A node is a machine that is a part of the Jenkins environment. The Jenkins Pipeline executes on a node block that is usually a part of the Scripted Pipeline syntax.

**Stage 🡺** Stage in a Jenkins Pipeline consists of a unique subset of tasks such as Build, Test, Deploy, etc. The Stage block is used by many plugins for providing the visualization of Jenkins status (and progress).

**Step 🡺** Step is a single task that tells Jenkins what exactly needs to be done. For example, setting an environment variable can be done in a step, executing a build command can also be a step. On the whole, a Jenkins Pipeline constitutes a series of steps.

**scripted pipeline syntax** 🡺

* Jenkins pipelines are traditionally written as scripted pipelines.
* Ideally, the scripted pipeline is stored in Jenkins webUI as a Jenkins file.
* The end-to-end scripted pipeline script is written in Groovy.
* It requires knowledge of Groovy programming as a prerequisite.
* Jenkinsfile starts with the word node.
* Can contain standard programming constructs like if-else block, try-catch block, etc.

Sample Scripted Pipeline

node {

    stage('Stage 1') {

        echo 'hello'

    }

}

**Declarative Pipeline** 🡺

The Declarative Pipeline subsystem in Jenkins Pipeline is relatively new, and provides a simplified, opinionated syntax on top of the Pipeline subsystems.

* The latest addition in Jenkins pipeline job creation technique.
* Jenkins declarative pipeline needs to use the predefined constructs to create pipelines. Hence, it is not flexible as a scripted pipeline.
* Jenkinsfile starts with the word pipeline.

**Jenkins declarative pipeline should be the preferred way to create a Jenkins job as they offer a rich set of features**, come with less learning curve & no prerequisite to learn a programming language like Groovy just for the sake of writing pipeline code.

We can also validate the syntax of the Declarative pipeline code before running the job. It helps to avoid a lot of runtime issues with the build script.

pipeline {

agent any

stages {

stage('Welcome Step') {

steps {

echo 'Welcome to LambdaTest'

}

}

}

}

**jenkins pipeline syntax :**

**Pipeline**

* It is a user-defined framework that includes all the processes like create, check, deploy, etc. In a Jenkinsfile, it’s a list of all the levels. All of the stages and steps within this block are described. This is the fundamental block to the syntax of a declarative pipeline.

pipeline  {

}

**Node**

* A node is a system running a complete workflow. It’s an integral part of the syntax of the scripted pipeline.

|  |  |
| --- | --- |
| { | node { |

Some standard sections are available to both declarative and scripted pipelines. These are:

**Agent**

* An agent is described as a directive that can run multiple builds using just one Jenkins instance. This feature helps spread the workload to various agents and execute multiple projects within Jenkins’s single instance. It instructs Jenkins to assign the builds to an executor.
* A single agent may be defined for a whole Jenkins pipeline, or different agents may be assigned to execute each stage within a pipeline. Some of the most commonly used Agent parameters are:

**Any**

* Runs the stage pipeline on any available agent.

**None**

* This parameter is added to the root of the pipeline. It means that there is no global agent for the entire pipeline, and each stage must define its own agent.

**Label**

* Performs on the labeled agent the pipeline/stage.

**Docker**

* This parameter uses a docker container as a pipeline execution environment or as a specific level. For example, the docker can be used to pull an image of Ubuntu. This image can now be used to run multiple commands as an execution environment.

pipeline {

agent {

docker {

image 'ubuntu'

}

}

}

**Stages**

* This section includes all of the work that needs to be completed. The work is defined in the form of stages. Within this Directive, there may be more than one level. Each stage executes a particular task.

pipeline {

agent any

stages {

stage ('Build') {

}

stage ('Test') {

}

stage ('QA') {

}

stage ('Deploy') {

}

stage ('Monitor') {

}

}

}

**Steps**

* Within a stage block, the pipeline can be described as a series of steps. Such steps are performed in sequence for the execution of a level. Within a Steps guideline, there must be at least one step.

pipeline {

agent any

stages {

stage ('Build') {

steps {

echo

'Running build phase. '

}

}

}

}

**SONARCUBE**

Software testing a apart of software development lifecycle, its aim is to ensure that the code to be deployed is of high quality with no bugs and no logical errors

Developer 🡺 Tester 🡺 Deployment

**Testing classification🡺**

Testing Type:

1. Manual :
2. Automatic :

Testing Methods :

1. Static:
2. Dynamic :

Testing Approaches:

1. Black Box : Tester doesn’t know application stary
2. White Box : Tesre knows application internal structure
3. Gray Box :

Testing Levels :

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance testing

Dynamic testing :

Developer 🡺 code 🡺 testing 🡺 if no errors

.