* **What is Devops:**
* DevOps=Development + Operations.
* DevOps is a culture or process.
* To automate the application, build, and deployment process.
* By using DevOps culture, we can establish collaboration between development and operational teamwork.
* By using DevOps culture, we can speed up application delivery.
* DevOps is a set of Tools.
* DevOps is the combination of best practices and tools which is designed to increase an organization company's ability to deliver applications/ projects and services faster than traditional software development processes.
* This speed enables organizations to better serve their customers and compete more effectively in the market.
* **Benefits of DevOps:**
* Free and open source
* Faster development
* Rapid delivery
* Improved collaboration
* Automated processes.
* **Jenkins features:**
* Open-source platform for implementing devops pipelines
* Cross platform
* Cost savings
* Plugin ecosystem-Jenkins supports thousands of plugins
* Increased productivity
* **Why it becomes more popular:**
* Reduced risk
* Better quality code
* Ready to ship
* Systematic versioning
* Code quality trend analysis
* Time to market
* Reduced cost.
* **Popular Devops CI/CD tools**:
* Jenkins
* Circle CI
* Bamboo
* Travis and many more.
* **Different phases in devops:**
* Plan
* Code
* Build
* Test
* Release
* Deploy
* Operate
* Monitor
* **Three important points in Devops:**
* Continuous integration
* Continuous deployment
* Continuous delivery.

**Continuous integration:**

Continuous integration is a devops software development practice where developers frequently merge their code changes into a central repository, after which automated builds and tests run.

**Continuous deployment:**

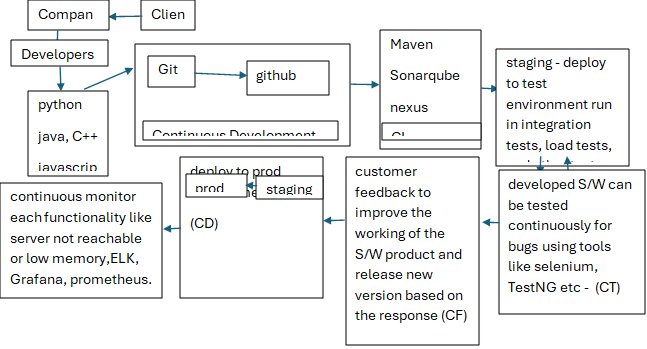
The code built in the CI process is automatically deployed to a higher environment, usually a staging/test environment.

**Continuous delivery:**

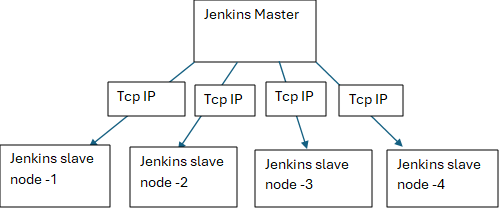
Code that passes functional and system tests is deployed to production.

Typically, a test lead or the release manager triggers this manually.

* **DevOps workflow:**

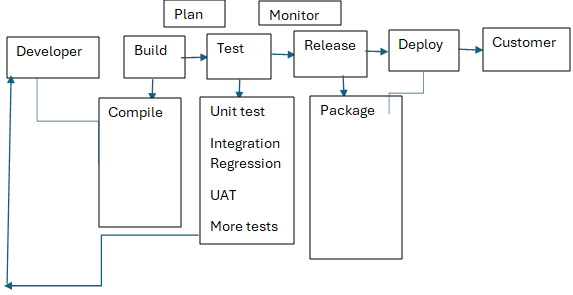


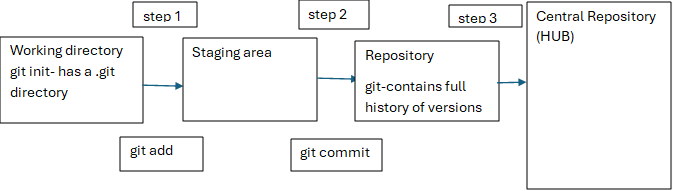
* **Jenkins deployment architecture:**
* Jenkins uses the master and slave architecture to manage distributed builds.
* Master node distribute load to slave nodes



* **Jenkins pipeline as code:**
* **What is pipeline in Jenkins:**
* In Jenkins, a pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.
* Pipelines automate the delivery process, reducing manual effort and improving efficiency.

**Note:** build pipeline plugin.

* **Jenkins pipelines vs job DSL:**
* They both have the capability to write all your CI/CD in code
* The difference is in implementation in Jenkins
* Jenkins job DSLs creates new jobs based on the code you write
* The Jenkins pipelines is a job type, you can create a Jenkins pipeline job that will handle the build/test/deployment of one project.
* **Note**: Job DSL plugin
* **CI/CD within the SDLC:**
* 
* **Scaling Jenkins:**
* Jenkins use master slave architecture
* The main aim is distributing build workloads across multiple slave nodes, reducing the load on the master node.
* **Git:**
* Git is a distributed version control system that enables multiple developers to collaborate on software projects efficiently.
* It is free and open source.
* It was created by Linux Torvalds in 2005.
* Git helps you keep track of code changes.
* It is version control system means it is also known as source control is the practice of tracking and managing changes to software code.
* designed to handle everything from small to very large projects with speed and efficiency.
* It is also called source control system.
* Tracking and managing changes to source code overtime.
* **Config:**
* The purpose of configuration is identifying who made changes
* Username and email are crucial.
* Git config --global user.name “enter name”
* Git config --global user.email “enter email”
* To see the config details git config --global --list.
* **Goal of git:**
* Speed
* Data integrity
* Support for distributed
* Nonlinear workflows.
* **Two types of version control systems:**
* Centralized version control system
* Distributed version control system.
* **How to create GitHub account, create repository, and clone a repository:**
* Go to browser
* Enter GitHub sign in
* Give email, username and password
* Account will be created.
* **Create repository in GitHub:**
* Click on new
* Enter repository name
* Click on public or private
* Click on README file
* Click on create repository
* Repository created.
* **Clone a repository:**
* Git clone <Repository URL>
* Clone means copy the files from the remote repository to our local machine.
* **Stages in git:**
* **There are three stages in git:**



* Working area/untracked area (git add <filename>)
* Staging area/ tracked area (git commit -m “message”)
* Local repository. (files will see in local repository).
* **Git pull:**
* In git pull is a command that fetches changes from a remote repo and merges them into the local repository.
* **Git push:**
* Push means that sends committed changes from a local repository to remote repository.
* **Git log:**
* To see the commit history.

Git log --oneline(to see all commits)

Git show <commitid> (to see single commit in detailed view)

* Head: it means lastest commit happened on your system.
* **Branches:**
* Branching in git is the process of creating a new, independent line of development in a repository. This allows developers to work on new features, bug fixes, or other changes without affecting the main codebase.
* You can merge a branch into another branch using a pull request.
* **Different types of branches:**
* **Master:**
* The master branch which will create contains production code all development code is merge into master after release.
* **Develop:**
* This branch contains the pre-production code inside develop branch we have create number of branches.
* **Feature:**
* a branch created for a specific feature or set of features.
* **Release:**
* a branch created for a specific release or version of the software.
* **Bug fix:**
* it fixes the bugs.
* **Create branches:**
* **Git branch <branch name> -** it creates new branch
* **Git branch-** it shows all the branches under the repository.
* **Git checkout <branch name>-** switch from one branch to another.
* **Git push --set-upstream origin <branch name>-** push a new branch to a remote repository and upstream tracks the information for branch.
* **Git branch -d <branch name> -** it deletes the branch
* **Git diff branch name..branch name-** it shows difference between two branches**.**
* **How to create pull request and merge one branch into another:**
* Merge data from feature branch to develop branch
* Select feature branch
* Select base branch is develop and compare branch is feature.
* Click on pull request
* Click on create pull request
* Add new description
* Click on create pull request
* Click on merge pull request
* Click on confirm merge.
* **Conflicts and how to resolve it:**
* For example, two branches created:
* Branch one is testforconflict
* Branch two is develop
* When you tried to merge testforconflict into develop branch conflict will come because it will try to merge all the files from testforconflict. The file name would be same so no conflict. Content is different than conflict will come.
* **How to resolve this conflict:**
* We can download one software that is **"tortoisegit".**
* By using tortoisegit we can resolve the conflict
* Open tortoisegit
* Right click on the conflict
* One popup window open
* Click on use this whole file.
* **Git clean:**
* The git clean command is used to removed untracked files and directories from a repository.
* **Undoing changes on existing file**
* This command will discard any changes made to the specified file and revert it to its previous state.

Git checkout -- <filename>

* **Git revert:**
* The git revert command is used to create a new commit that undoes the changes made in a previous commit.

Git revert <commit id>

Note: it works like undo.

* **Git reset**:
* It is used to reset the current branch head to a specified state and optionally reset the index and working tree to match.

Git reset --soft <commit id> (files not removed) remove the commit id from the logs from the history.

Git reset --hard <commit id> (files removed)

* **Git fetch:**
* Git fetch only downloads new data from a remote repository.

Git pull = git fetch +git merge

* **Git merge**:
* The git merge command is used to integrate changes from one branch into another
* **Meaning of cherry pick:**
* Cherry pick is a command that allows you to apply a specific commit from one branch to another.

Git cherrypick <commit id>

* **Git tags:**
* Git tags are used to mark specific points in a repository’s history, making it easier to reference and track changes.
* Tags are used to mark a commit stage as relevant. we can tag a commit for future reference.
* **BitBucket:**
* It is a remote and free service for source code management.
* Bitbucket is a web-based source control hosting service.
* The projects the use git revision control system.
* Bitbucket offers both commercial plans and free plans.
* **Note**: install bitbucket branch source plugin
* **Maven core concepts:**
* **Pom.xml file:**
* Pom.xml stands for project object model and .xml means Xtensible markup language
* Must be present in the projects root directory
* Root element <project>
* Contains everything needed to build a project using maven.
* **Maven coordinates:**
* The first few elements of pom.xml file makes maven coordinates.
* The first few elements groupid, artifactid, version from the coordinates.
* It marks a specific place in a repository and hence called coordinates.
* **Maven goals:**
* A goal is an action that maven performs in a phase.
* A plugin is a collection of goals
* **Maven plugins:**
* Need to information more about single plugin mvn help:describe -Dplugin=compiler
* **Maven lifecycle:**
* Mvn clean
* Mvn validate
* Mvn compile
* Mvn test
* Mvn package
* Mvn verify
* Mvn install
* Mvn deploy
* **maven repositories:**
* There are three repositories:
* Remote
* Central
* Local
* **Dependency management:**
* This section contains main libraries the project depends on
* **Convention over configuration:**
* Every artifact of a project has its own place.
* Source files-src/main/java-source code
* Test files-src/test/java-test code
* Packaged files-target directory-war/jar/ear
* Pom.xml-stored in root directory
* **Eclipse IDE for maven installation:**
* Go to browser
* Enter Eclipse download
* Eclipse IDE for java EE developers
* Click download 64bits in windows
* Unzip it
* Open it
* Go to file
* Click on import
* Click on existing maven projects
* Next
* Click on project location
* Finish.
* After completion of import project name will be displayed on the left side.
* **Compiler plugin:**
* Mvn help:describe -Dplugin=compiler (it gives all the details of the compiler plugin and all the goals description etc)
* **Email notifications:**
* **How to integrate email notifications into your jobs**
* Go to manage Jenkins
* Click on configure system
* Go to smtp server
* Enter smtp.gmail.com
* Click use SSL checkbox
* Give smtp port 465
* Click on use smtp authentication
* Username and password
* Click on test configuration by sending test email by sending test email recipient
* click on test configuration.

**Note**: Email extension plugin

* **Slack integration:**
* Slack is a chat and collaboration tool, comparable with Atlassian’s HipChat.
* It's a new and popular communication tool, that is much better than using skype or older enterprise chat tools for collaboration.
* **Note**: Download –slack notification plugin
* **API integration**:
* Sometimes you want to integrate an API, but there is no plugin available.
* Another solution is to use functionality in the jenkins pipelines to do http requests.
* Note: Http request plugin.
* **SonarQube integration:**
* SonarQube continuously inspects your software project on code quality.
* **Note:** install sonarqube scanner for jenkins plugin.
* **Authentication:**
* The process or action of verifying the identity of a user and process.
* Basically, verifying the credentials of the user: the username and password.
* **Onelogin.com**
* Note:Install SAML plugin(security Assertion Markup Language)
* **Authorization:**

**Meaning of authorization:**

Authorization in Jenkins refers to the process of controlling access to Jenkins resources, such as jobs, views, and nodes, based on user permissions.

* Important points:

1. How to create a job in jenkins
2. How to configure
3. How to integrate with github,maven, sonarqube.
4. How to create slave nodes
5. How to create views.
6. How to create pipeline in Jenkins.

**Pratical:**

**Branches:**

After creating one repository automatically one default branch will be created that is main branch

I created one new branch by using this command (**git branch <master>**)

This branch was created in local repository . We want to push this branch into remote remote repository(git push –-set-upstream origin master) go to github we will see newly created branch.

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (main) $

git branch

* main

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (main) $ ls -ltr total 35 -rw-r--r-- 1 Administrator 197121 34 Apr 3 12:42 README.md -rw-r--r-- 1 Administrator 197121 415 Apr 3 12:42 Hello.class -rw-r--r-- 1 Administrator 197121 1153 Apr 3 15:42 pom.xml -rw-r--r-- 1 Administrator 197121 20775 Apr 3 21:58 'java\_project steps and output.docx' drwxr-xr-x 1 Administrator 197121 0 Apr 4 20:36 target/ -rw-r--r-- 1 Administrator 197121 115 Apr 13 00:33 Hello.java drwxr-xr-x 1 Administrator 197121 0 Apr 22 07:26 java\_sample\_project\_1/

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (main) $

git branch master

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (main) $

git branch

* Main
* Master

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (main) $

git push --set-upstream origin master

(master branch added to the github account)

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (master) $ touch file1

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (master) $

ls

Hello.class Hello.java README.md file1 'java\_project steps and output.docx' java\_sample\_project\_1/ pom.xml target/

Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (master) $

vi file1 Administrator@DESKTOP-I16IA9O MINGW64 ~/Desktop/sample\_java\_project (master) $ cat file1 this is my first file

**Push from feature branch to develop branch**

Go to github account

Click on pull request

Click new pull request

Select base-develop and compare-feature(i merge feature branch into develop branch)

Create pull request

Give commit

Create pull request

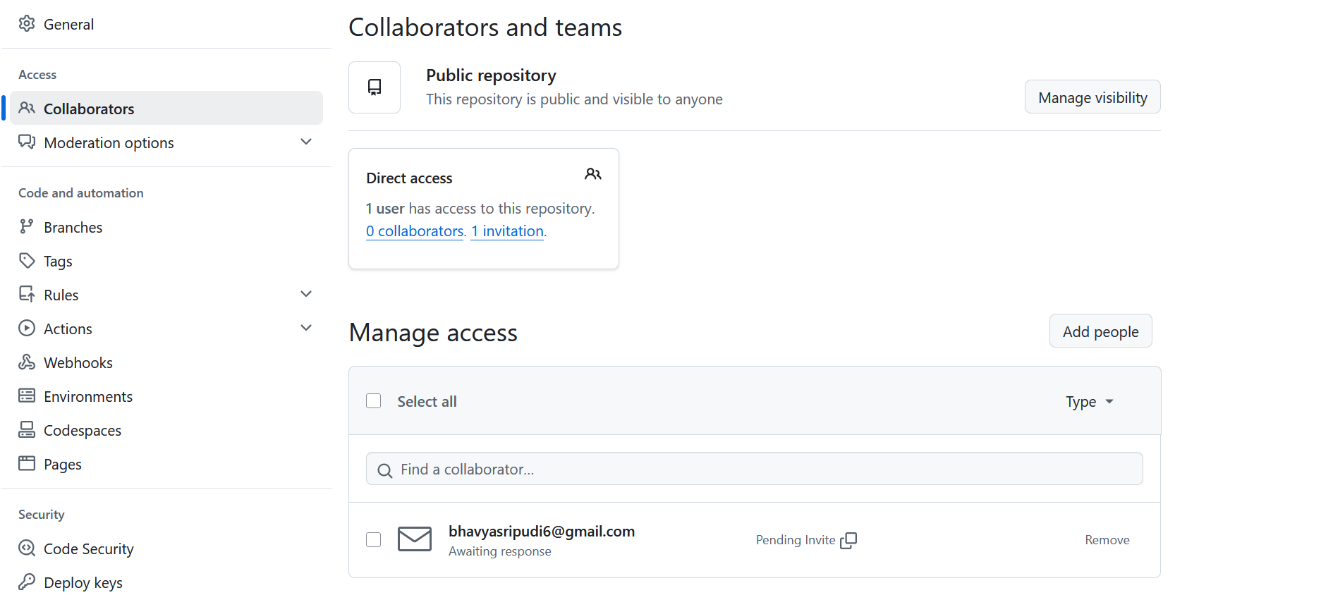
Merge pull request

Confirm merge.

Note : if u want to delete the branch (git branch –-delete <branch name>)

It deletes only in local machine. We need to push it will be deleted in remote repository.

(git push –-delete origin <branch name>

* **Add reviewer:**
* Settings
* Branches
* Add branch ruleset
* Give branch name pattern(develop)
* Require a pull request before merging
* Required approvals(1)
* Create
* Asking password(enter github password) first step done
* Settings
* Collaborators
* Manage access
* Add people
* Enter email
* Add <email>

Go to mail and click view invitation

Click accept invitation.

