

TABLE OF CONTENTS

S.NO	CONTENTS	PAGE NO
1	INTRODUCTION	2
2	ABSTRACT	3
3	SYSTEM ANALYSIS	4
	3.1 EXISTING SYSTEM	
	3.2 PROPOSED SYSTEM	
	3.3 SYSTEM REQUIREMENTS	
4	3-TIER ARCHITECTURE	7
5	TOOLS	17
6	CODING	46
7	SCREENSHOTS	68
8	CONCLUSION	71
9	BIBLOGRAPHY	72

INTRODUCTION

Scope:

Provides end to end continuous Integration and continuous deployment by using DevOps Tools.

Application defines student to provide entire information and which stores in DB for better analysis of data.

Provides security using 3-tier architecture by queueing the load in the web server queue when the server is bombarded with load.

Objective:

Deploying application with 3-tier and bring the application to live using cloud infrastructure

Jenkins CI CD tool for end-to-end automation and any build failures will be detect at the starting stages

Assigning the DNS to the Load Balancer IP with the Freenom subscription.

For https security to the domain created open ssl certificates to make the domain https.

Analysing the logs at the Load balancer level, web server level and app server level for better understanding the application flow.

ABSTRACT

The main aim of our project is to build and deploy application in 3-tier architecture. Our main motto of this project is to develop an application with full end-to-end automation using ci/cd build process and deploying war file into tomcat servers using DevOps tools.

DevOps is a conceptual framework for reintegrating and development and operations of informations systems. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment, and operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

As the project based on automation, it shows less errors and reduces time compared to manual process.

SYSTEM ANALYSIS

EXISTING SYSTEM

- The current system of the project is manually deploying the builds to DEV, QA, PROD environments.
- Because of manual deployments there is a chance for human errors.
- The manual process will associated with a lot of problems such as missing student details in our student registration form, Time wastage etc.

PROPOSED SYSTEM

- The proposed system is end-to-end automation with DevOps tools.
- This system makes the user to work with agile methodology more efficiently.
- It also reduces the work of developers to troubleshoot the deployment issues and focus on the code.
- The main aim of this project is to build and deploy application in 3-tier architecture with automatic CI/CD build process.

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

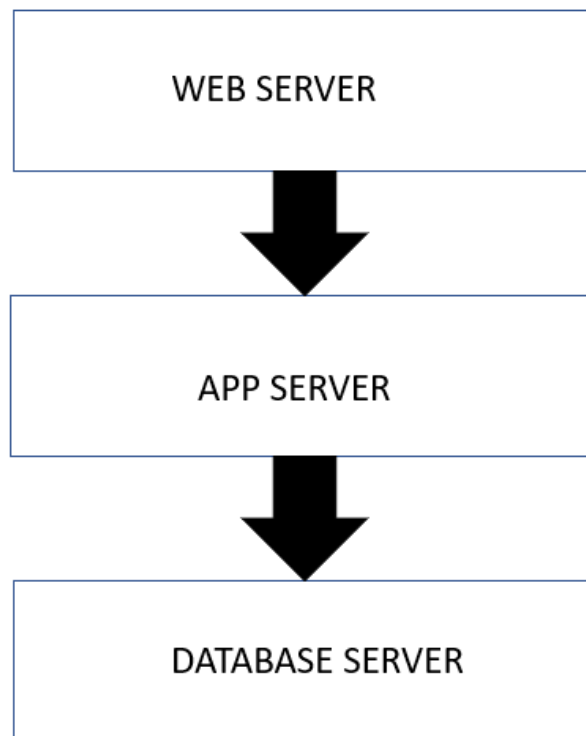
- PROCESSOR : 2VCU'S
- RAM : 4GB
- HDD : 10GB

SYSTEM REQUIREMENTS:

- OPERATING SYSTEM : LINUX/WINDOWS
- FRONTEND CODE : HTML
- BACKEND CODE : JAVA
- SERVER : APACHE TOMCAT 8
- DATABASE : MARIADB

3-TIER ARCHITECTURE

3-TIER ARCHITECTURE:



3-tier architecture consists of three layers. They are:

- Web layer
- Application layer
- Database layer

WEB LAYER:

- Web server's basic job is to accept user requests and send responses to those requests.

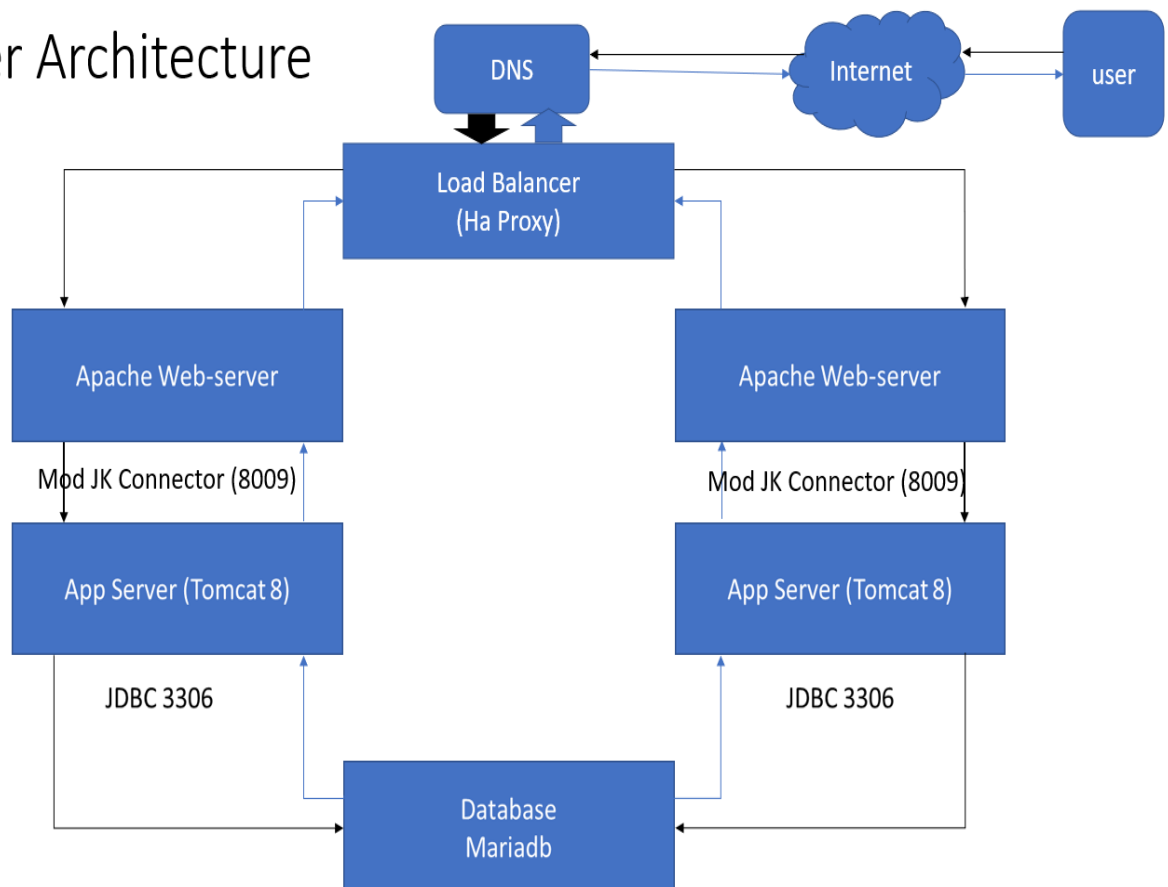
APP LAYER:

- App layer takes the request from web layer and process the request

DATABASE LAYER:

- Database stores the data given by the user and also extract the data when user requires.

3-Tier Architecture



Apache Web Server:

- Apache Web server is one of the most popular web servers.
- It was developed by the Apache Software Foundation open source community.
- Most of the web server machines run the Apache web server.
- According to google almost 50% of web servers around the world runs on Apache web server.

Apache Tomcat Server:

- Apache Tomcat is an open source web server.
- It was developed by Apache Software Foundation. ASF executes Java servlets and JavaServer pages including JSP coding.

- Tomcat is developed and maintained by an open collaboration of developers.
- It is available in both binary and source versions from the Apache Web site.
- It requires a JRE (Java Runtime Enterprise) that conforms to JRE 1.1 or later.
- It can be used as its own internal web server or with other web servers like Apache, Netscape Enterprise Server, Microsoft Personal Web Server and Microsoft Internet Information Server (IIS).

Features of Apache Server:

- Apache server is a free and an open source web server.
- It can be installed on all operating systems like Linux, Windows, Unix, FreeBSD, Solaris, Mac OS X etc.
- It is a powerful, flexible, HTTP/1.1 compliant web server.
- This server is highly configurable and extensible with third party modules.
- It provides complete source code and comes with an unrestricted license.
- Apache supports some of the capabilities like CGI (Common Gateway Interface) and SSI (Server Side Includes), URL redirection, loading modules support, user authentication, proxy caching abilities etc.
- It is also widely used in the software companies.

MariaDB Database:

MariaDB is a popular fork of MySQL created by MySQL's original developers. It grew out of concerns related to MySQL's acquisition by Oracle. It offers support for both small data processing tasks and enterprise needs. It aims to be a drop-in replacement for MySQL requiring only a simple uninstall of MySQL and an install of MariaDB.

MariaDB is a relational database management system. It stores data in various tables. Primary keys and foreign keys are used to establish relationship between these tables.

Key Features of MariaDB:

The important features of MariaDB are –

- All of MariaDB is under GPL, LGPL, or BSD.
- MariaDB includes a wide selection of storage engines, including high-performance storage engines, for working with other RDBMS data sources.
- MariaDB uses a standard and popular querying language.
- MariaDB runs on a number of operating systems and supports a wide variety of programming languages.
- MariaDB offers support for PHP, one of the most popular web development languages.
- MariaDB offers Galera cluster technology.
- MariaDB also offers many operations and commands unavailable in MySQL, and eliminates/replaces features impacting performance negatively.

LOAD BALANCER:

Load balancing is a key component of highly-available infrastructures commonly used to improve the performance and reliability of web sites, applications, databases and other services by distributing the workload across multiple servers.

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool.

HAPROXY serves as a load balancer which is widely used in the companies.

DNS:

The domain name system (DNS) connects URLs with their IP address. With DNS, it's possible to type words instead of a string of numbers into a browser, allowing people to search for websites and send emails using familiar names. When you search for a domain name in a browser, it sends a query over the internet to match the domain with its corresponding IP. Once located, it uses the IP to retrieve the website's content. Most impressively, this whole process takes just milliseconds.

Connectors:

Connectors are used to establish connection between the servers.

Mod_jk connector:

The mod_jk connector is an Apache HTTPD module that allows HTTPD to communicate with Apache Tomcat instances over the AJP protocol. The module is used in conjunction with Tomcat's AJP Connector component.

Mod-JK Configuration:

```
cd /etc/apache2/  
vim workers.properties  
worker.list=worker1  
worker.worker1.type=ajp13  
worker.worker1.host=<IP-ADDRESS-OF-TOMCAT-SERVER>  
worker.worker1.port=8009
```

```
vim /etc/apache2/mods-available/jk.conf  
JkWorkersFile /etc/apache2/workers.properties  
JkLogFile /var/log/apache2/mod_jk.log  
vim /etc/apache2/sites-enabled/000-default.conf  
JkMount /studentapp* worker1
```

Java database connectivity (JDBC):

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database, and is oriented toward relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.

JDBC Connection:

Update context.xml for DB Connections

```
vim /var/lib/tomcat8/conf/context.xml
```

Add the following config just before the last line

```
<Resource name="jdbc/TestDB" auth="Container"
type="javax.sql.DataSource"
        maxActive="50" maxIdle="30" maxWait="10000"
        username="student" password="student@1"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://<IP ADDRESS OF DB
SERVER>:3306/studentapp"/>
```

Scaling:

Scaling is the process of increasing or decreasing the capacity of the system by changing the number of processes available to service requests. Scaling out a system provides additional capacity, while scaling in a system reduces capacity. Scaling is also a critical part of configuring a deployment for high availability.

Methods of adding more resources for a particular application fall into two broad categories:

- Scale vertically (scale up)
- Scale horizontally (scale out)

Scale up:

“Scale up” is when you upgrade a machine to a more powerful machine (e.g. faster CPU, faster GPU, engine with more HP, etc...) to get more processing power.

To scale vertically (or scale up) means to add:

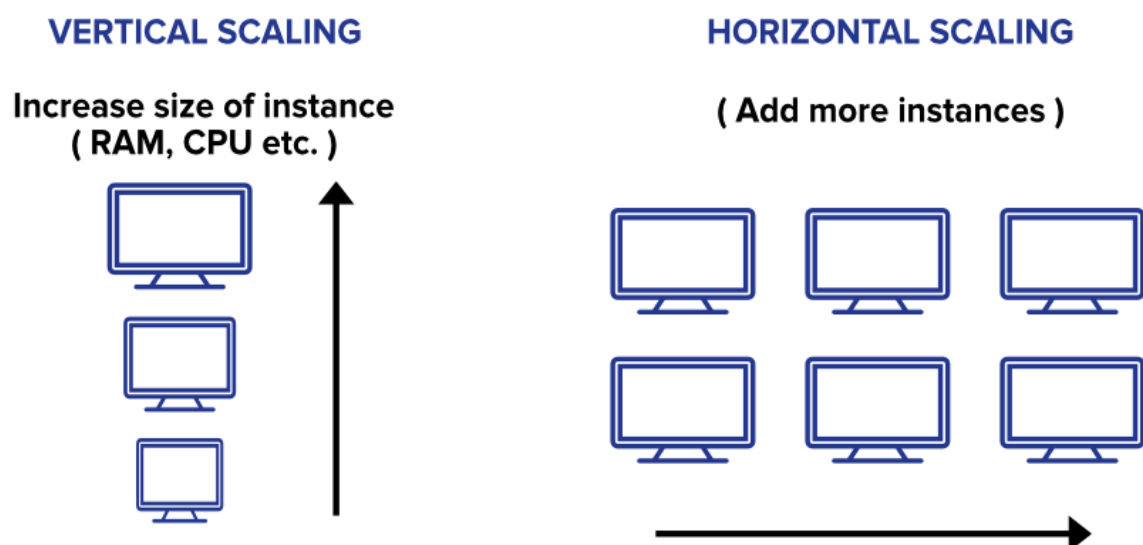
- resources (addition of CPUs or memory)
- or process (addition of the same process)

to a single node in a system.

Scale out:

“Scale out” is when you increase the number of processing machines (computers, processors, servers, etc) to increase processing power.

To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application.



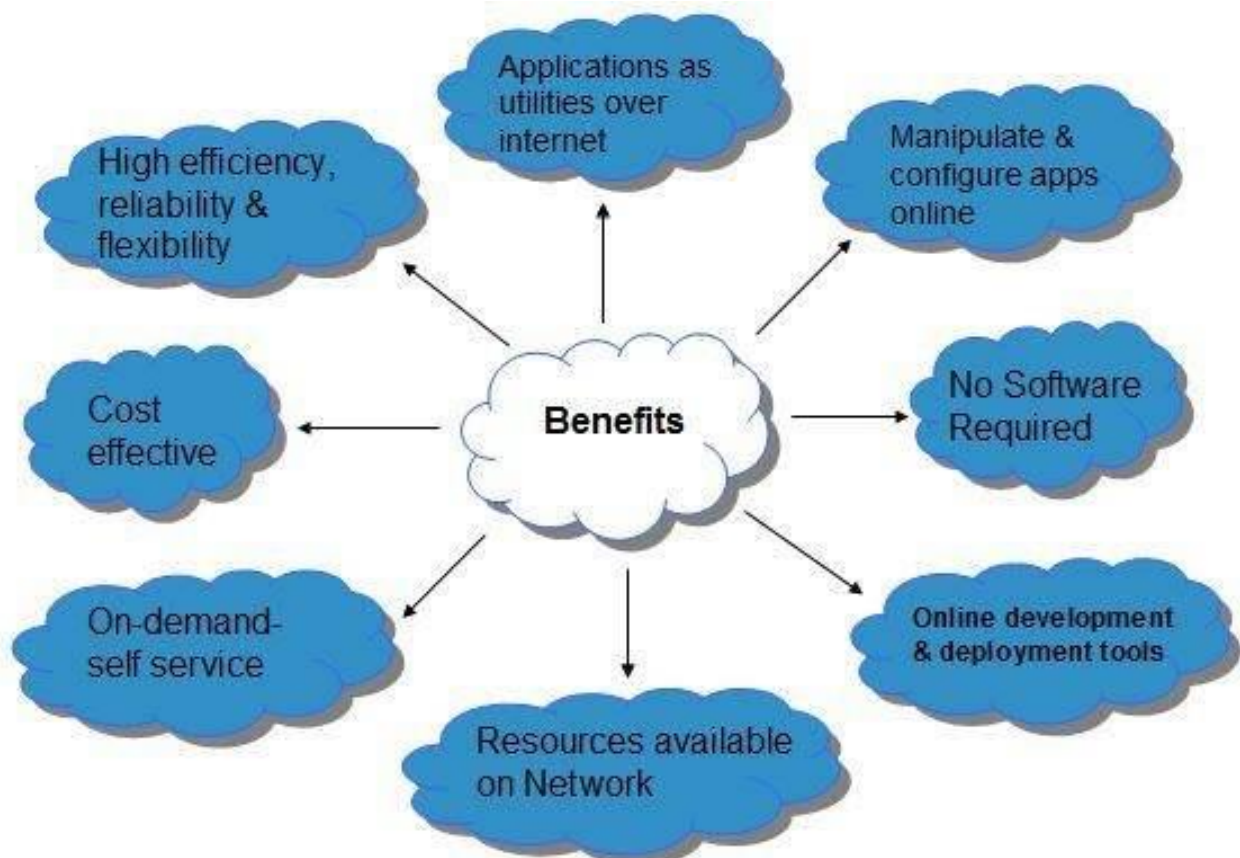
Cloud:

The term **Cloud** refers to a **Network** or **Internet**. In other words, we can say that Cloud is something, which is present at remote location. Cloud can provide services over public and private networks, i.e., WAN, LAN or VPN.

Cloud Computing provides us means of accessing the applications as utilities over the Internet. It allows us to create, configure, and customize the applications online.

Cloud Computing refers to **manipulating**, **configuring**, and **accessing** the hardware and software resources remotely. It offers online data storage, infrastructure, and application.

Benefits:



All though there are so many cloud based technologies, these 3 are widely used clouds. They are:

- AWS (Amazon Web Services)
- Microsoft Azure
- Google Cloud Platform (GCP)

Note:

In our project we have used Google Cloud Platform.

GCP Servers:

Filter VM instances							Columns
<input type="checkbox"/> Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect	
<input type="checkbox"/> appserver	us-central1-a			10.128.0.4 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> dbserver	us-central1-a			10.128.0.5 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> haproxy	us-east1-b			10.142.0.4 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> jenkins	us-central1-a			10.128.0.2 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> nexus	us-central1-a			10.128.0.3 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> web1	us-east1-b			10.142.0.2 (nic0)	None	SSH ▾	⋮
<input type="checkbox"/> web2	us-east1-b			10.142.0.3 (nic0)	None	SSH ▾	⋮

TOOLS

GIT

Git is an **open-source distributed version control system**.

It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

Git is easy to learn, and has fast performance. It is superior to other **Source Code Management** tools like Subversion, CVS, Perforce, and ClearCase.

Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.

GITHUB

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.

It offers both **distributed version control and source code management (SCM)** functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

Features of GitHub:

GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open-source projects and codes of various programming languages.

Some of its significant features are as follows.

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting
- Track and assign tasks
- Conversations

Benefits of GitHub:

GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

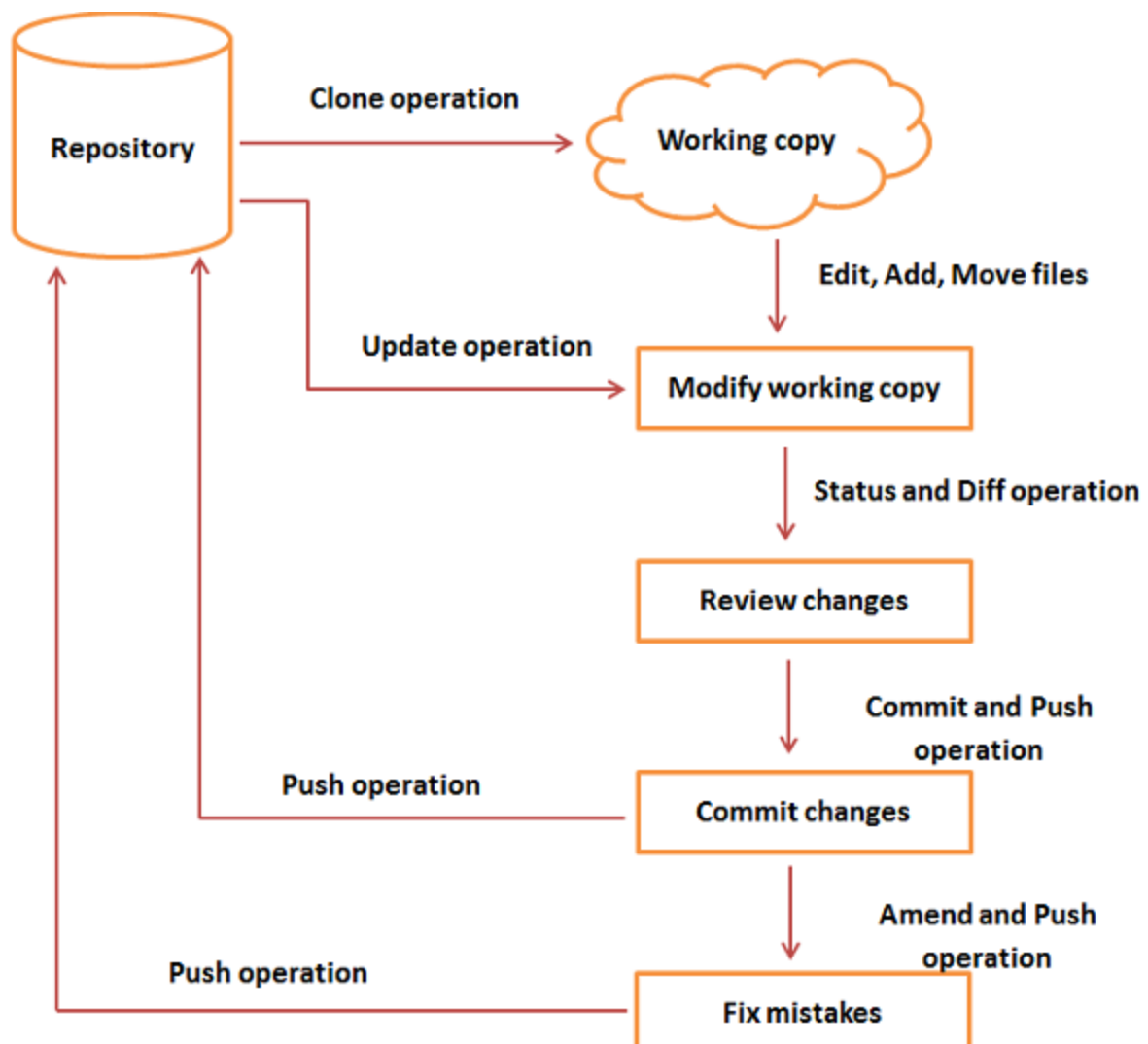
- It is easy to contribute to open-source projects via GitHub.
- It helps to create an excellent document.
- It allows your work to get out there in front of the public.
- You can track changes in your code across versions.

Git - Life Cycle:

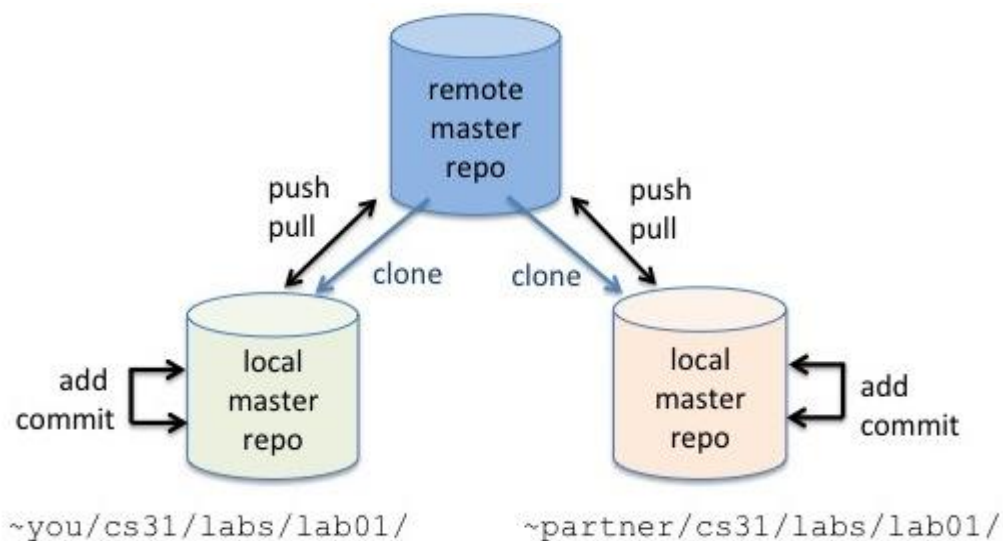
General workflow is as follows –

- You clone the Git repository as a working copy.
- You modify the working copy by adding/editing files.
- If necessary, you also update the working copy by taking other developer's changes.
- You review the changes before commit.
- You commit changes. If everything is fine, then you push the changes to the repository.
- After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.

Shown below is the pictorial representation of the work-flow.

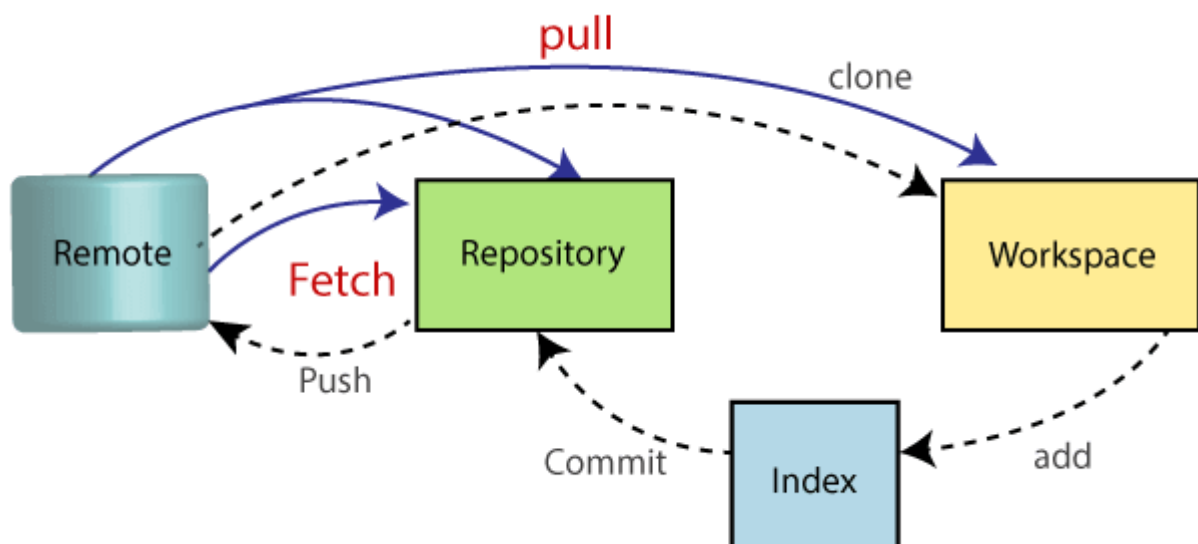


Git-Commands:



Git Pull:

The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The **git pull command** is used to pull a repository.



The pull command is used to access the changes (commits) from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a

collection of the fetch and merges command. First, it fetches the changes from remote and combined them with the local repository.

The syntax of the git pull command is given below:

Syntax:

```
git pull <option> [<repository URL><refspec>...]
```

In which:

<option>: Options are the commands; these commands are used as an additional option in a particular command. Options can be **-q** (quiet), **-v** (verbose), **-e**(edit) and more.

<repository URL>: Repository URL is your remote repository's URL where you have stored your original repositories like GitHub or any other git service. This URL looks like:

<https://github.com/saisiva2502/studentapp>

Git Force Pull:

Git force pull allows for pulling your repository at any cost. i.e., used for overwriting the files.

Step1: Use the git fetch command to download the latest updates from the remote without merging or rebasing.

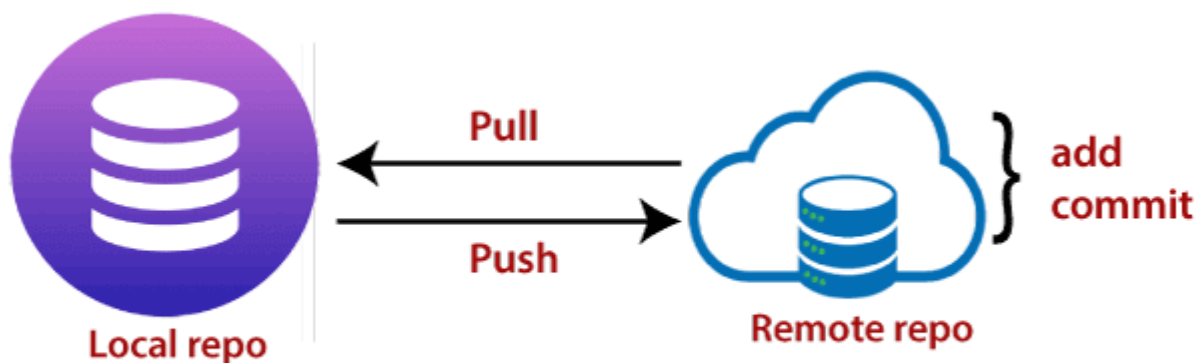
```
git fetch -all
```

Step2: Use the git reset command to reset the master branch with updates that you fetched from remote. The hard option is used to forcefully change all the files in the local repository with a remote repository.

```
git reset -hard <remote>/<branch_name>  
git reset-hard master
```

Git Push:

The "git push" command is used to push into the repository. The push command can be considered as a tool to transfer commits between local and remote repositories.



If we do not specify the location of a repository, then it will push to default location at **origin master**.

The basic syntax is given below:

```
git push<option>[<Remote URL> <branch name> <refspec>]
```

Git Push Origin Master:

Git push origin master is a special command-line utility that specifies the remote branch and directory. When you have multiple branches and directory, then this command assists you in determining your main branch and repository.

Generally, the term **origin stands** for the remote repository, and master is considered as the main branch. So, the entire statement "**git push origin master**" pushed the local content on the master branch of the remote location.

Syntax:

```
git push origin master
```

Working Directory and Staging Area or Index:

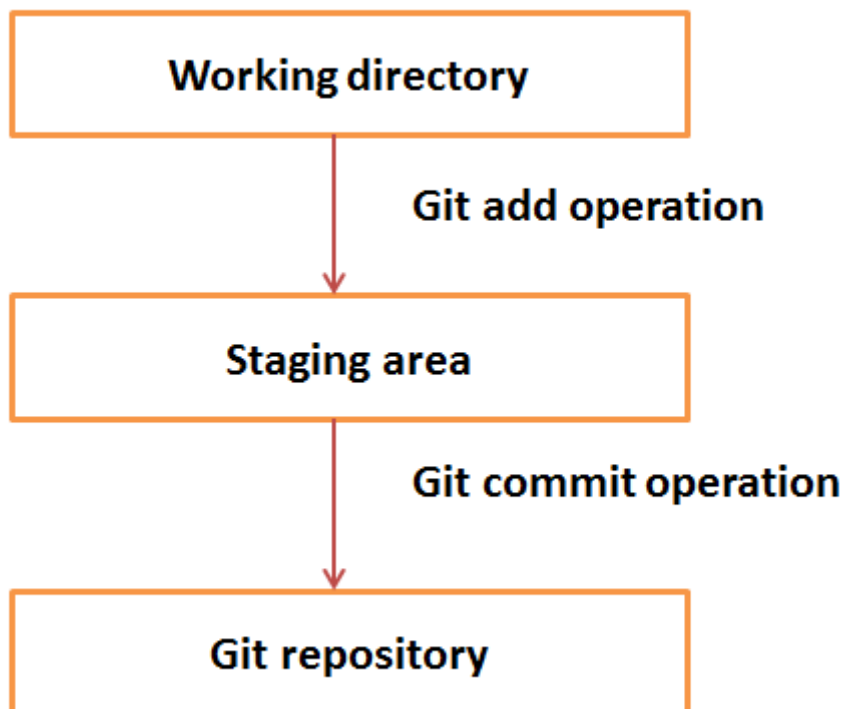
The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn't track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.

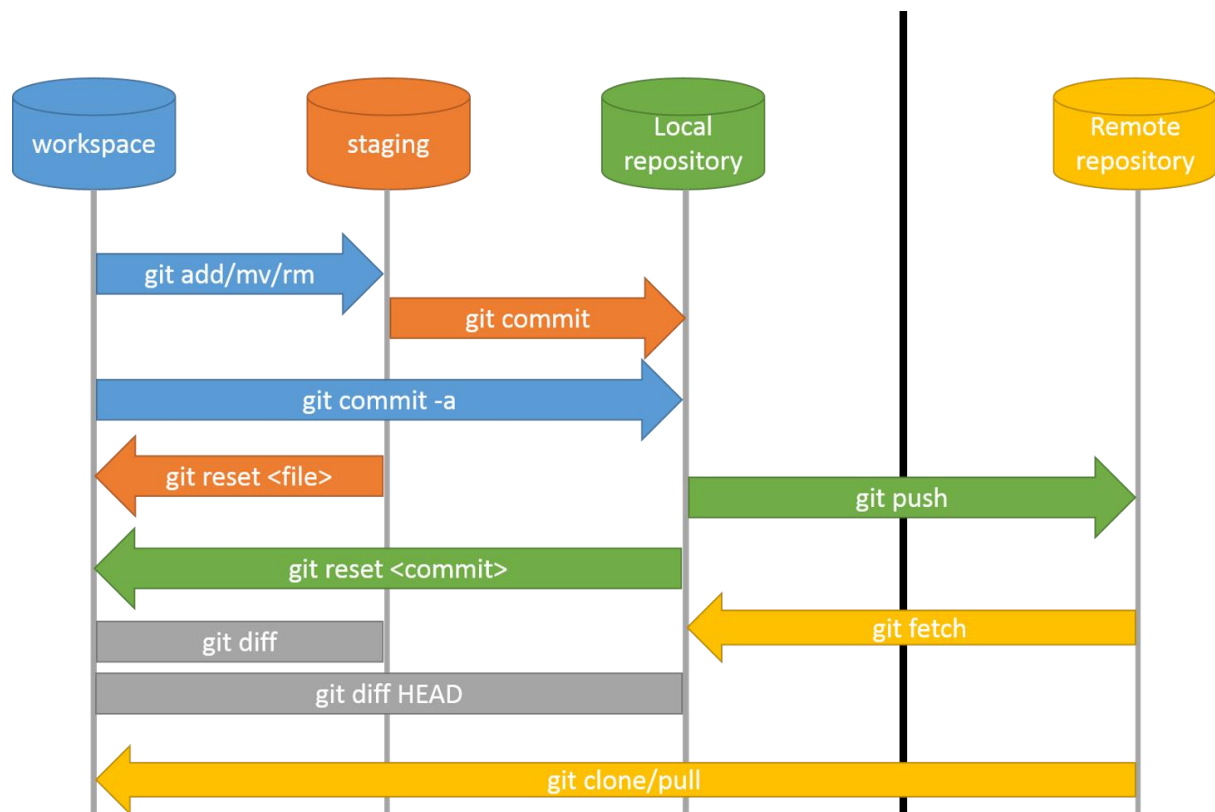
Let us see the basic workflow of Git.

Step 1 – You modify a file from the working directory.

Step 2 – You add these files to the staging area.

Step 3 – You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.





Git Fetch:

The "**git fetch**" **command** is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

Syntax:

`git fetch < repository URL >`

Git Init:

The `git init` command is the first command that you will run on Git. The `git init` command is used to create a new blank repository. It is used to make an existing project as a Git

project. Several Git commands run inside the repository, but init command can be run outside of the repository.

The git init command creates a .git subdirectory in the current working directory. This newly created subdirectory contains all of the necessary metadata. These metadata can be categorized into objects, refs, and temp files. It also initializes a HEAD pointer for the master branch of the repository.

Syntax:

```
git init
```

Git Add:

The git add command is used to add file contents to the Index (Staging Area). This command updates the current content of the working tree to the staging area. It also prepares the staged content for the next commit. Every time we add or update any file in our project, it is required to forward updates to the staging area.

The git add command is a core part of Git technology. It typically adds one file at a time, but there some options are available that can add more than one file at once.

Git add command is a straight forward command. It adds files to the staging area. We can add single or multiple files at once in the staging area. It will be run as:

Syntax:

```
git add <File name>
```

To add all the files from the repository, run the add command with **-A** option. We can use '.' Instead of **-A** option. This command will stage all the files at a time.

Syntax:

```
git add -A (or)  
git add .
```

Git Commit:

It is used to record the changes in the repository. It is the next command after the git add. Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area. A commit command is used to fetch updates from the staging area to the repository.

The staging and committing are co-related to each other. Staging allows us to continue in making changes to the repository, and when we want to share these changes to the version control system, committing allows us to record these changes.

Syntax:

```
git commit -am "Commit message."
```

The commit command also provides **-a** option to specify some commits. It is used to commit the snapshots of all changes. This option only consider already added files in Git. It will not commit the newly created files.

The **-m** option of commit command lets you to write the commit message on the command line. This command will not prompt the text editor.

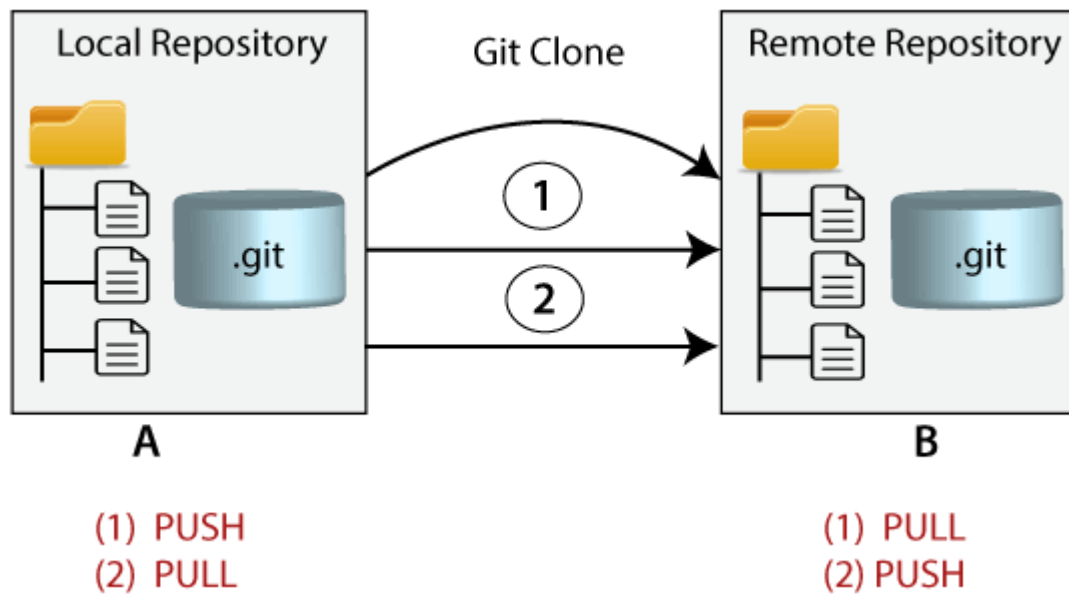
Git Clone:

The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the **origin**.

Syntax:

```
git clone <repository URL>
```



Git Reset:

The term reset stands for undoing changes. The git reset command is used to reset the changes.

Syntax:

```
git reset --hard
```

Git log:

Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.

Git log command is one of the most usual commands of git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head.

Syntax:

```
git log
```

Git Diff:

Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc. It compares the different versions of data sources. The version control system stands for working with a modified version of files. So, the diff command is a useful tool for working with Git.

Syntax:

```
git diff --staged
```

To check the staged changes, run the git diff command along with **--staged** option.

Git Status:

The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.

Mostly, it is used to display the state between **Git Add** and **Git commit** command. We can check whether the changes and files are tracked or not. To check the status, open the git bash, and run the status command on your directory.

Syntax:

```
git status
```

MAVEN

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following –

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

Objective:

The primary goal of Maven is to provide developer with the following –

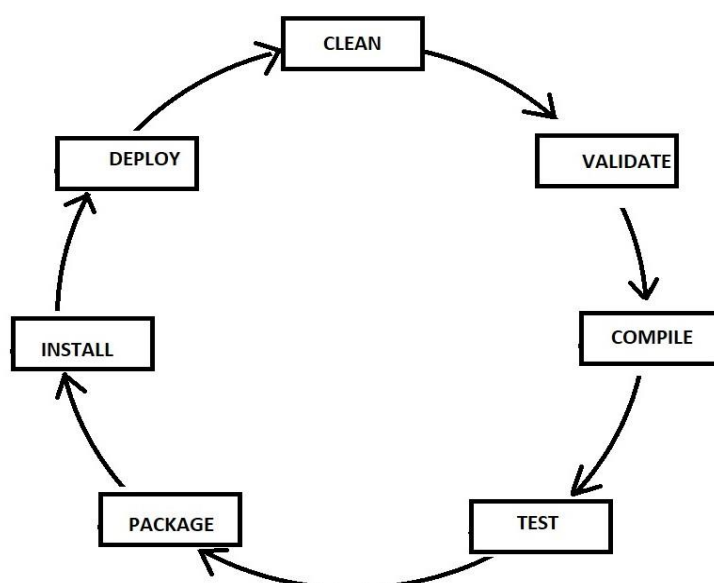
- A comprehensive model for projects, which is reusable, maintainable, and easier to comprehend.
- Plugins or tools that interact with this declarative model.

Maven project structure and contents are declared in an xml file, pom.xml, referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system.

Features of Maven:

- Simple project setup that follows best practices.
- Consistent usage across all projects.
- Dependency management including automatic updating.
- A large and growing repository of libraries.
- Extensible, with the ability to easily write plugins in Java or scripting languages.
- Instant access to new features with little or no extra configuration.

Maven - Build Life Cycle:



MAVEN COMMANDS:

CLEAN:

Clears the target directory into which maven normally builds your project.

Syntax:

```
mvn clean
```

Validate:

Validates if the project is correct and if all necessary information is available.

Syntax:

```
mvn validate
```

Compile:

Source code compilation is done in this phase.

Syntax:

```
mvn compile
```

Test:

Tests the compiled source code suitable for testing framework.

Syntax:

```
mvn test
```

Package:

This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.

Syntax:

```
mvn package
```


Install:

This phase installs the package in local/remote maven repository.

Syntax:

```
mvn install
```

Deploy:

Copies the final package to the remote repository.

Syntax:

```
mvn deploy
```

Maven pom.xml file:

POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configuration that can be specified in the POM are following –

- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

Before creating a POM, we should first decide the project **group** (groupId), its **name** (artifactId) and its version

as these attributes help in uniquely identifying the project in repository.

It should be noted that there should be a single POM file for each project.

- All POM files require the project element and three mandatory fields: groupId, artifactId, version.

Elements of maven pom.xml file:

For creating the simple pom.xml file, you need to have following elements:

project:

It is the root element of pom.xml file.

modelVersion:

It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.

groupId:

It is the sub element of project. It specifies the id for the project group.

artifactId:

It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.

version:

It is the sub element of project. It specifies the version of the artifact under given group.

POM.XML FILE:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.jdevs</groupId>

  <artifactId>studentapp</artifactId>

  <version>2.5-SNAPSHOT</version>

  <packaging>war</packaging>

  <build>

    <sourceDirectory>src</sourceDirectory>

    <plugins>

      <plugin>

        <artifactId>maven-compiler-plugin</artifactId>

        <version>3.3</version>

        <configuration>

          <source>1.8</source>

          <target>1.8</target>

        </configuration>

      </plugin>

      <plugin>

        <artifactId>maven-war-plugin</artifactId>

        <version>2.6</version>

        <configuration>

          <warSourceDirectory>WebContent</warSourceDirectory>

          <failOnMissingWebXml>>false</failOnMissingWebXml>

        </configuration>

      </plugin>

    </plugins>

  </build>

</project>
```

```
</configuration>
</plugin>
</plugins>
</build>
<distributionManagement>
  <repository>
    <id>deployment</id>
    <name>Internal Releases</name>
    <url>http://10.128.0.10:8081/repository/student-
rel/</url>
  </repository>
  <snapshotRepository>
    <id>deployment</id>
    <name>Internal Snapshot Releases</name>
    <url>http://10.128.0.10:8081/repository/student-
snap/</url>
  </snapshotRepository>
</distributionManagement>
<dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
  </dependency>
</dependencies>
</project>
```

JENKINS

Jenkins is an open source automation tool written in Java programming language that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies.

Jenkins **builds** and **tests** our software projects which 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 us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

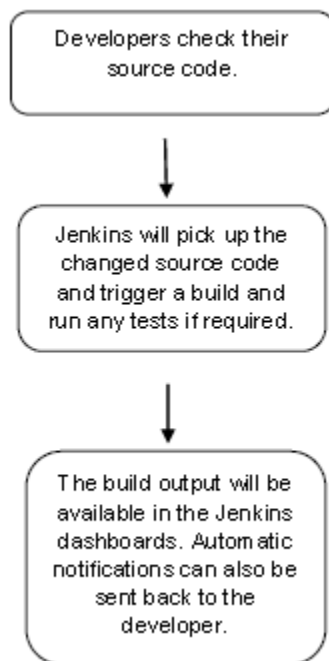
Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

For example: If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script
- Archive a build result
- Running software tests

Work Flow:

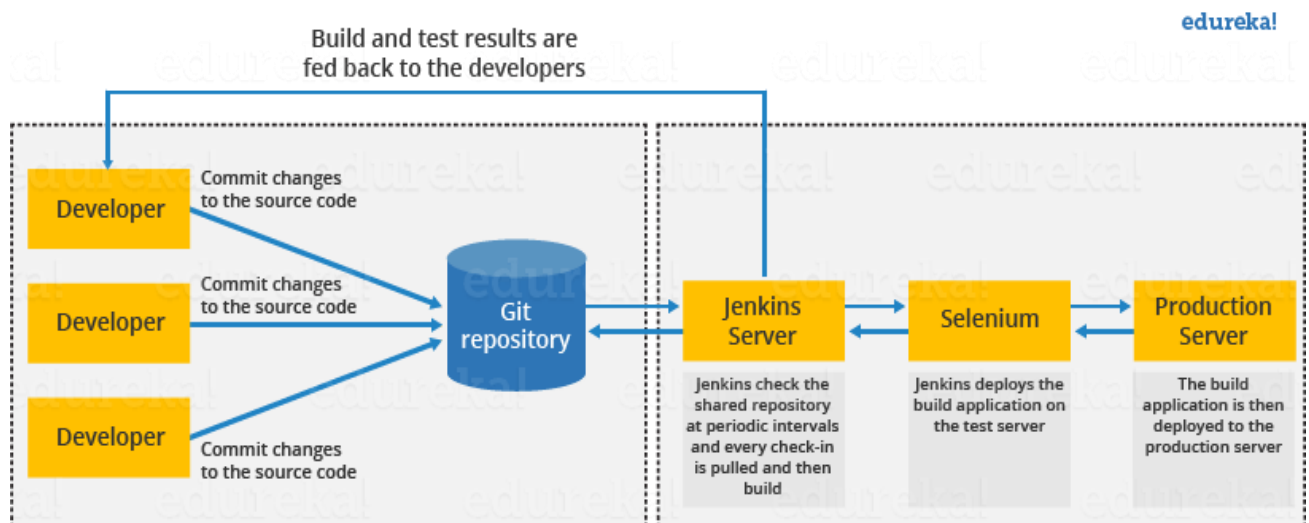


Continuous Integration:

Continuous Integration (*CI*) is a development practice in which the developers are needed to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

Generic flow diagram of Continuous Integration with Jenkins:

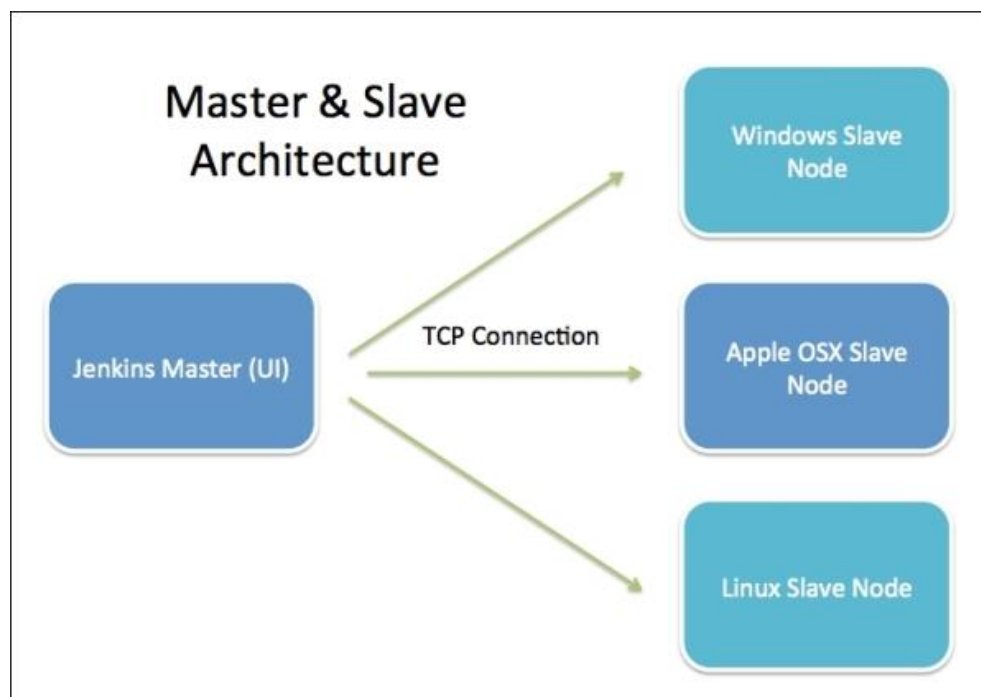


Jenkins Architecture:

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server



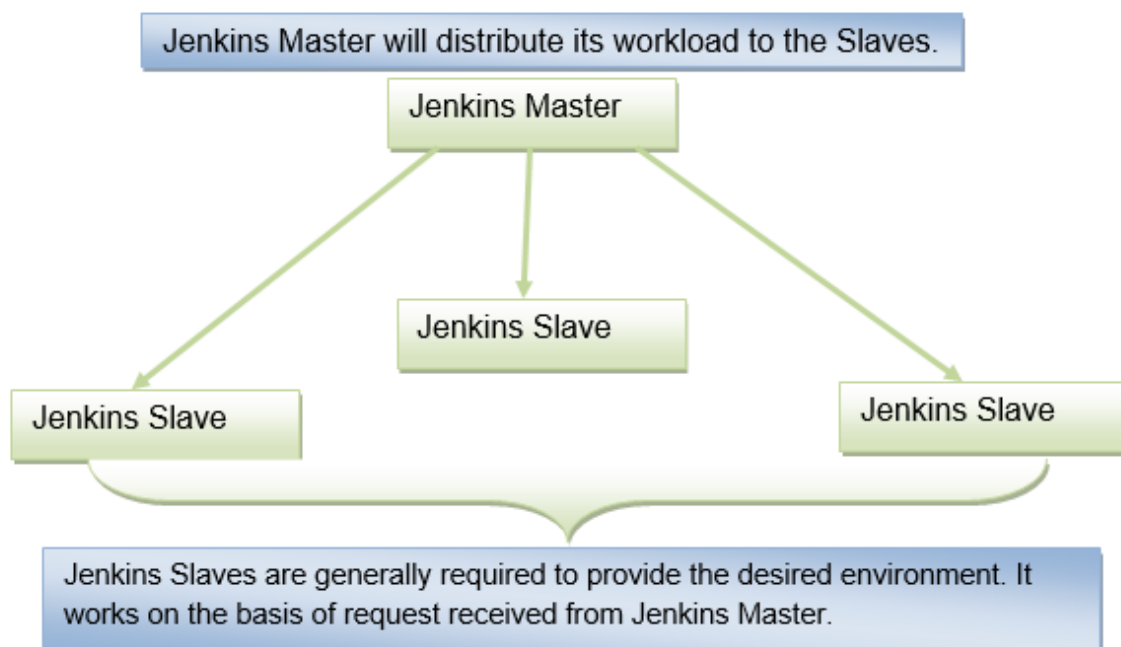
Jenkins Master:

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

Jenkins Slave:

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.



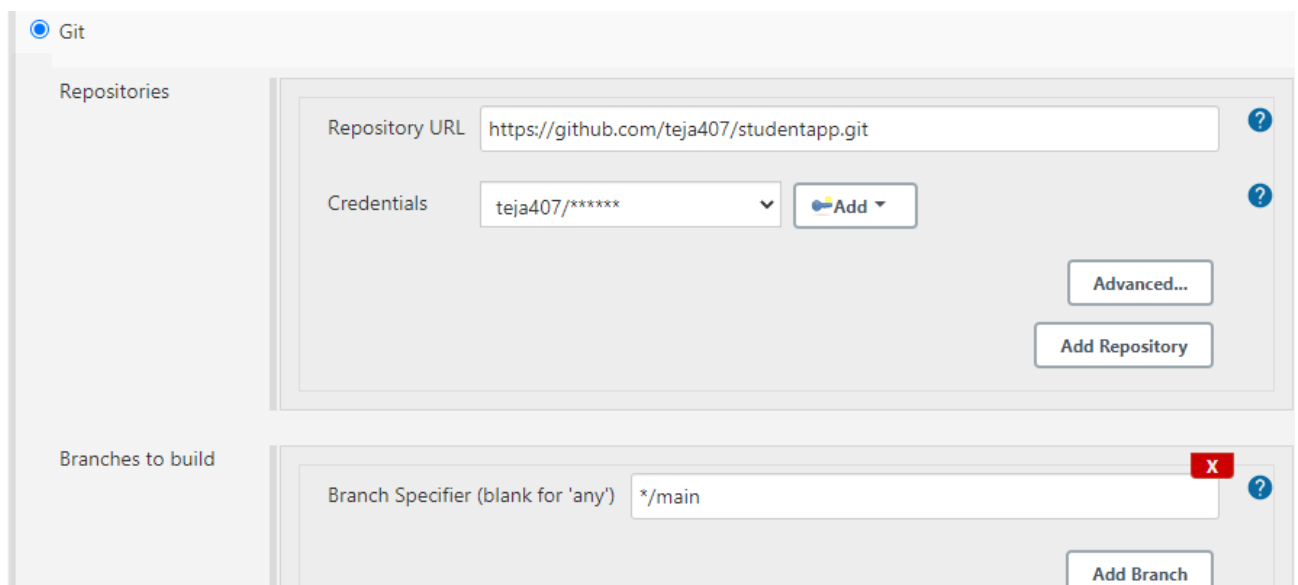
The above diagram is self explanatory. It consists of a Jenkins Master which is managing three Jenkins Slaves.

GitHub Setup for Jenkins:

Jenkins is a CI (Continuous Integration) server and this means that it needs to check out source code from a source code repository and build code. Jenkins has outstanding support for various source code management systems like Subversion, CVS etc. To do the GitHub setup, first we need to install "Git Plugin".

Integrating Jenkins with GitHub:

- First create a new job in Jenkins, open the Jenkins Dashboard and click on "create new jobs".
- Now enter the item name and select the job type. For example, item name is "**test**" and job type is "**Freestyle project**". Click on **OK**.
- Once you click OK, the page will be redirected to its project configuration.
- Now, under the "Source Code Management" you will see the Git option, if your **Git** plugin has been installed in Jenkins.
- Enter the Git repository URL on the "Repository URL" option to pull the code from GitHub.



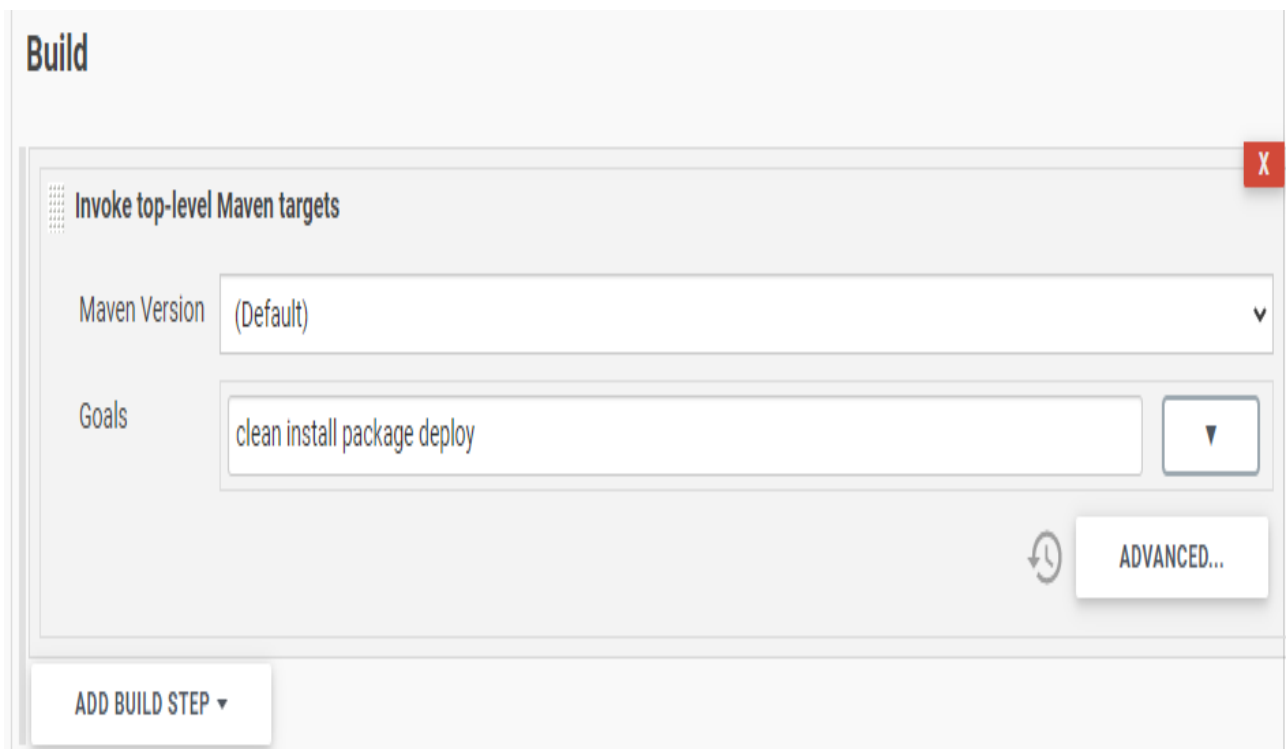
The screenshot shows the Jenkins configuration page for the Git plugin. On the left, there is a sidebar with a radio button selected for "Git". Below it, the "Repositories" section is visible. The main configuration area contains a "Repository URL" field with the value "https://github.com/teja407/studentapp.git", a "Credentials" dropdown menu showing "teja407/*****", and an "Add" button. Below these are "Advanced..." and "Add Repository" buttons. The "Branches to build" section is also visible, with a "Branch Specifier (blank for 'any')" field containing "*/main" and an "Add Branch" button. A red "X" icon is present next to the branch specifier field.

Maven Setup for Jenkins:

To do the Maven setup, first we need to install "Maven Plugin".

Integrating Jenkins with Maven:

- Open already created "Test" project, then click on configure option.
- Now, below the "Source Code Management" you will see the Build option, In the Build Triggers section, there are multiple options, select the Maven one, If your Maven plugin has installed.



The screenshot shows the 'Build' section of the Jenkins configuration interface. It features a 'Build' header, a 'Invoke top-level Maven targets' section with a red close button, and a 'Goals' field containing 'clean install package deploy'. Below the goals field is an 'ADVANCED...' button with a clock icon. At the bottom left, there is an 'ADD BUILD STEP' button with a dropdown arrow.

Build

Invoke top-level Maven targets

Maven Version (Default) ▼

Goals clean install package deploy ▼

⌚ ADVANCED...

ADD BUILD STEP ▼

Tomcat Setup for Jenkins:

First we need to install "Tomcat Server" and "Deploy plugin" in Jenkins.

Integrating Jenkins with Tomcat:

- Open already created "Test" project, then click on configure option.
- Now, below the "Source Code Management" you will see the Post Build option, In the Post Build Triggers section, select the Deploy plugin, If your Deploy plugin has installed.

The screenshot shows the 'Post-build Actions' configuration page in Jenkins. The 'Deploy war/ear to a container' option is selected. The configuration includes the following fields:

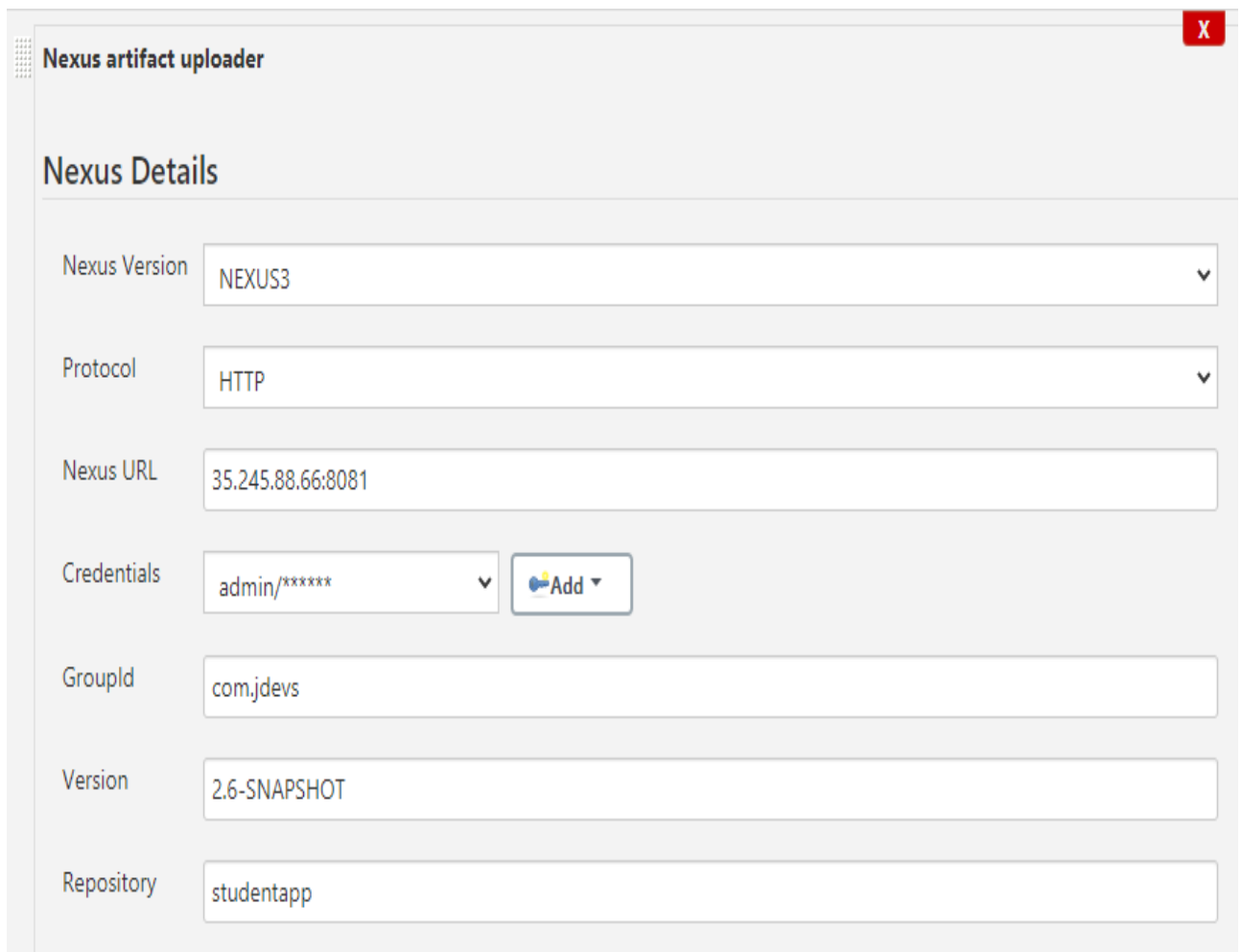
- WAR/EAR files:** mavenrepo-master/target/studentapp-2.6-SNAPSHOT.war
- Context path:** student
- Containers:**
 - Tomcat 8.x Remote:**
 - Credentials:** tomcat/***** (tomcat-cred) [Add]
 - Tomcat URL:** http://34.86.153.144:8080 [Advanced...]
 - [Add Container]
- Deploy on failure:** ☐

Nexus Setup for Jenkins:

First we need to install "Nexus artifact uploader" in Jenkins.

Integrating Jenkins with Nexus:

- Open already created "Test" project, then click on configure option.
- Now, below the "Source Code Management" you will see the Build option, In the Build Triggers section, there are multiple options, select the Nexus one, If your **Nexus artifact uploader** plugin has installed.



The screenshot shows the "Nexus artifact uploader" configuration page in Jenkins. The page has a title bar with a red close button (X). Below the title bar is a section titled "Nexus Details". The form contains the following fields:

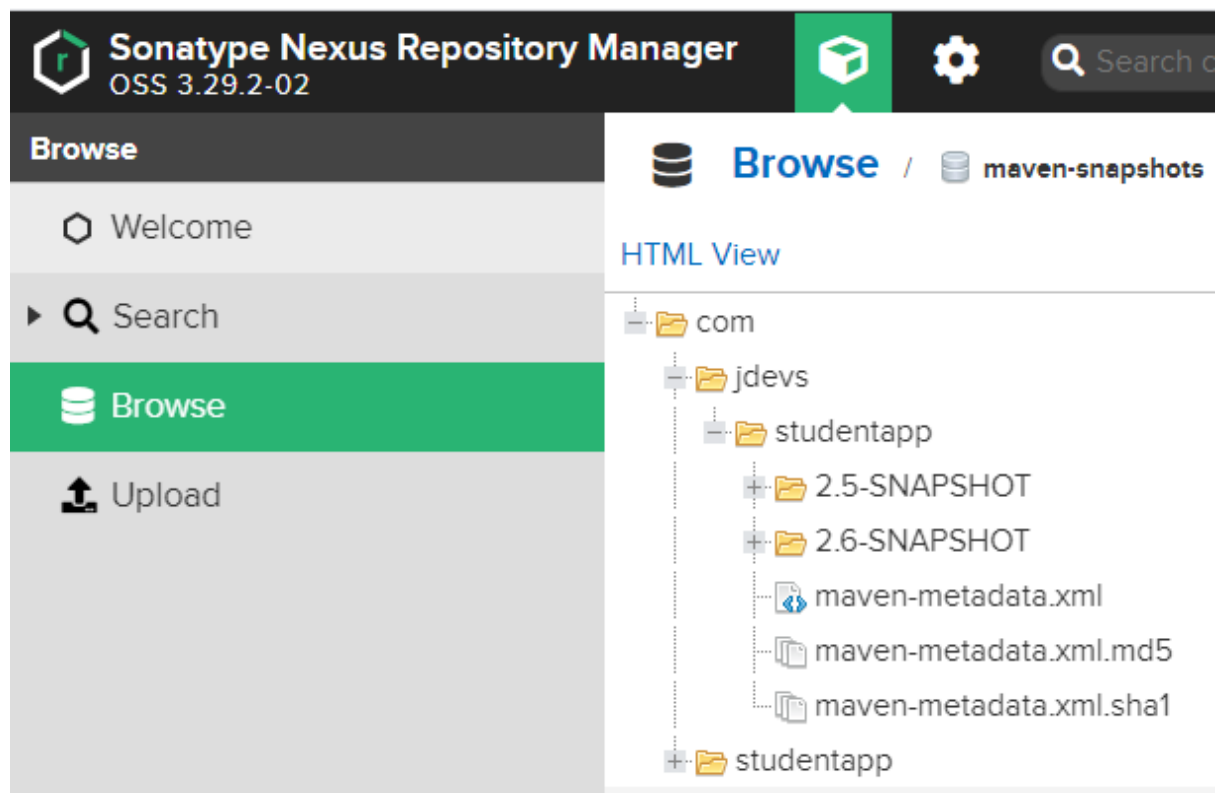
- Nexus Version: A dropdown menu with "NEXUS3" selected.
- Protocol: A dropdown menu with "HTTP" selected.
- Nexus URL: A text input field containing "35.245.88.66:8081".
- Credentials: A dropdown menu with "admin/*****" selected, and an "Add" button with a plus icon.
- GroupId: A text input field containing "com.jdevs".
- Version: A text input field containing "2.6-SNAPSHOT".
- Repository: A text input field containing "studentapp".

NOTE:

First we need to install java, tomcat, GitHub, maven, nexus in Jenkins server.

NEXUS

Nexus is a repository manager, it stores "artifacts". It allows you to proxy, collect, and manage your dependencies so that you are not constantly juggling a collection of JARs. It makes it easy to distribute your software. Internally, you configure your build to publish artifacts to Nexus and they then become available to other developers. You get the benefits of having your own 'central', and there is no easier way to collaborate." With Nexus, developers can completely control access to, and deployment of, every artifact in an organization from a single location, making it easier to distribute software. It is most commonly used for hosting Apache Maven.



CODING

HOME.JSP

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">

<title>Display</title>

<style>
table#nat{
    width: 50%;
    background-color: #c48ec5;
}
</style>
</head>
<body>
<% String name = request.getParameter("fullname");
    String Addr = request.getParameter("address");
    String age = request.getParameter("age");
    String Qual = request.getParameter("qual");
    String Persent = request.getParameter("percent");
    String Year = request.getParameter("yop"); %>
<table id ="nat">
<tr>
    <td>Full Name</td>
```

```

        <td><%= name %></td>
</tr>
<tr>
        <td>Address</td>
        <td><%= Addr %></td>
</tr>
<tr>
        <td>Age</td>
        <td><%= age %></td>
</tr>
<tr>
        <td>Qualification</td>
        <td><%= Qual %></td>
</tr>
<tr>
        <td>Percentage</td>
        <td><%= Percent %></td>
</tr>
<tr>
        <td>Year of Passout</td>
        <td><%= Year %></td>
</tr>
</table>
</body>
</html>

```


INDEX.JSP

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">

<title>User Data</title>

</head>

<style>
div.ex {
    text-align: right width:300px;
    padding: 10px;
    border: 5px solid grey;
    margin: 0px
}
</style>

<body>

    <h1>Student Registration Form</h1>

    <div class="ex">

        <form action="registrationController" method="post">

            <table style="width: 50%">

                <tr>

                    <td>Student Id</td>
```

```
        <td><input type="text" name="id"/></td>
    </tr>
<tr>
    <td>Student Full Name</td>
    <td><input type="text" name="fullname"/></td>
</tr>
<tr>
    <td>Student Permanent Address</td>
    <td><input type="text" name="address"/></td>
</tr>
<tr>
    <td>Student Age</td>
    <td><input type="text" name="age"/></td>
</tr>
<tr>
    <td>Student Qualification</td>
    <td><input type="text" name="qual"/></td>
</tr>
<tr>
    <td>Student Percentage</td>
    <td><input type="text" name="percent"/></td>
</tr>
<tr>
    <td>Year Passed</td>
    <td><input type="text" name="yop"/></td>
</tr>
```

```
/table>
```

```
<input type="submit" value="register"/>
```

```
</form>
```

```
</div>
```

```
</body>
```

```
</html>
```

NEWFILE.JSP

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

WEB.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
    <display-name>CustomWebApp</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

StudentDAO.java

```
package com.srk.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import vo.Student;
public class StudentDAO {
    /*public static Connection getConnection(){
        Connection con=null;
        try{
            Class.forName("com.mysql.jdbc.Driver");
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root");
        }catch(Exception e){System.out.println(e);}
        return con;
    }*/
    public static Connection getConnection() throws Exception
    {
        Connection conn=null;
        Context initContext = new InitialContext()
        Context envContext = (Context)
initContext.lookup("java:comp/env");
```

```

DataSource ds = (DataSource)
envContext.lookup("jdbc/TestDB");

        conn = ds.getConnection();
        return conn;
    }

    public static void main(String[] args) {

    }

    public static int saveStudent(Student std){
        int status=0;
        try{
            Connection con=StudentDAO.getConnection();
            PreparedStatement ps=con.prepareStatement("insert
into
Students(student_name,student_addr,student_age,student_qual,student_percent,student_year_passed) values
(?,?,?,?,?,?,?)");
            ps.setString(1,std.getStudentName());
            ps.setString(2,std.getStudentAddr());
            ps.setString(3,std.getAge());
            ps.setString(4,std.getQualification());
            ps.setString(5,std.getPercentage());
            ps.setString(6,std.getYearPassed());
            status=ps.executeUpdate();
            con.close();
        }catch(Exception ex){ex.printStackTrace();}
        return status;
    }

```

```

public static int updateStudent(Student std){
    int status=0;
    try{
        Connection con=StudentDAO.getConnection();
        PreparedStatement ps=con.prepareStatement("update
Students set
student_name=?,student_addr=?,student_age=?,student_qual
=?,student_percent=?,student_year_passed=? where
student_id=?");
        ps.setString(1,std.getStudentName());
        ps.setString(2,std.getStudentAddr());
        ps.setString(3,std.getAge());
        ps.setString(4,std.getQualification());
        ps.setString(5,std.getPercentage());
        ps.setString(6,std.getYearPassed());
        ps.setInt(7, std.getStudentId());
        status=ps.executeUpdate();
        con.close();
    }catch(Exception ex){ex.printStackTrace();}
    return status;
}

public static int deleteStudent(int stdId){
    int status=0;
    try{
        Connection con=StudentDAO.getConnection();
        PreparedStatement ps=con.prepareStatement("delete
from Students where student_id=?");
        ps.setInt(1,stdId);

```



```

        status=ps.executeUpdate();
        con.close();
    }catch(Exception e){e.printStackTrace();}
    return status;
}

public static Student getStudentById(int StdId){
    Student student=new Student();
    try{
        Connection con=StudentDAO.getConnection();
        PreparedStatement ps=con.prepareStatement("select
* from Students where student_id=?");
        ps.setInt(1,StdId);
        ResultSet rs=ps.executeQuery();
        if(rs.next()){
            student.setStudentId(rs.getInt(1));
            student.setStudentName(rs.getString(2));
            student.setStudentAddr(rs.getString(3));
            student.setAge(rs.getString(4));
            student.setQualification(rs.getString(5));
            student.setPercentage(rs.getString(6));
            student.setYearPassed(rs.getString(7));
        }
        con.close();
    }catch(Exception ex){ex.printStackTrace();}
    return student;
}

public static List<Student> getAllStudents(){

```

```

List<Student> students=new ArrayList<Student>();
try{
    Connection con=StudentDAO.getConnection();
    PreparedStatement ps=con.prepareStatement("select
* from Students");
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        Student student=new Student();
        student.setStudentId(rs.getInt(1));
        student.setStudentName(rs.getString(2));
        student.setStudentAddr(rs.getString(3));
        student.setAge(rs.getString(4));
        student.setQualification(rs.getString(5));
        student.setPercentage(rs.getString(6));
        student.setYearPassed(rs.getString(7));
        students.add(student);
    }
    con.close();
}catch(Exception e){e.printStackTrace();}
return students;
}
}

```

DeleteStudent.java

```
package com.srk.servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.srk.dao.StudentDAO;
@WebServlet("/deleteStudent")
public class DeleteStudent extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String sid=request.getParameter("stdId");
        int id=Integer.parseInt(sid);
        StudentDAO.deleteStudent(id);
        response.sendRedirect("viewStudents");
    }
}
```

EditStudent.java

```
package com.srk.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.srk.dao.StudentDAO;
import vo.Student;
@WebServlet("/editStudent")
public class EditStudent extends HttpServlet{
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<h1>Update Student</h1>");
        String sid=request.getParameter("stdId");
        int stdId=Integer.parseInt(sid);
        Student student=StudentDAO.getStudentById(stdId);
        out.print("<form action='editStudent2'
method='post'>");
        out.print("<table>");
```

```

        out.print("<tr><td></td><td><input type='hidden'
name='stdId'
value='"+student.getStudentId()+"'/></td></tr>");

        out.print("<tr><td>Full Name :</td><td><input
type='text' name='stdname'
value='"+student.getStudentName()+"'/></td></tr>");

        out.print("<tr><td>Address :</td><td><input
type='text' name='stdaddrs'
value='"+student.getStudentAddr()+"'/></td></tr>");

        out.print("<tr><td>Age :</td><td><input
type='text' name='stdage'
value='"+student.getAge()+"'/></td></tr>");

        out.print("<tr><td>Qualification :</td><td><input
type='text' name='stdqual'
value='"+student.getQualification()+"'/></td></tr>");

        out.print("<tr><td>Percentage :</td><td><input
type='text' name='stdpercent'
value='"+student.getPercentage()+"'/></td></tr>");

        out.print("<tr><td>Year of Passout
:</td><td><input type='text' name='stdyearpass'
value='"+student.getYearPassed()+"'/></td></tr>");

        out.print("<tr><td colspan='2'><input
type='submit' value='Edit & Save '/></td></tr>");

        out.print("</table>");
        out.print("</form>");
        out.close();
    }
}

```

RegistrationController.java

```
package com.srk.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.srk.dao.StudentDAO;
import vo.Student;
@WebServlet("/registrationController")
public class RegistrationController extends HttpServlet {
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("fullname");
        String Addr = request.getParameter("address");
        String age = request.getParameter("age");
        String Qual = request.getParameter("qual");
        String Persent = request.getParameter("percent");
        String Year = request.getParameter("yop");
```

```

        if(name.isEmpty()||Addr.isEmpty()||age.isEmpty()||Qual.i
sEmpty()||Persent.isEmpty()||Year.isEmpty())
        {
            RequestDispatcher rd =
request.getRequestDispatcher("index.jsp");
            out.println("<font color=red>Please fill all the
fields</font>");
            rd.include(request, response);
        }
        else
        {
            Student student = new Student();
            student.setStudentName(name);
            student.setStudentAddr(Addr);
            student.setAge(age);
            student.setQualification(Qual);
            student.setPercentage(Persent);
            student.setYearPassed(Year);
            int status=StudentDAO.saveStudent(student);
            if(status>0){
                response.sendRedirect("viewStudents");
            }else{
                out.println("Sorry! unable to save record");
            }
            out.close();
        }
    }}

```

SaveEditedStudent.java

```
package com.srk.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.srk.dao.StudentDAO;
import vo.Student;
@WebServlet("/editStudent2")
public class SaveEditedStudent extends HttpServlet{
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String sid=request.getParameter("stdId");
        int studentId=Integer.parseInt(sid);
        String studentName = request.getParameter("stdname");
        String studentAddrs = request.getParameter("stdaddrs");
        String studentAge = request.getParameter("stdage");
        String studentQual = request.getParameter("stdqual");
        String studentPercent = request.getParameter("stdpercent");
        String studentYearPass=request.getParameter("stdyearpass");
```



```
Student student = new Student();
    student.setStudentId(studentId);
    student.setStudentName(studentName);
    student.setStudentAddr(studentAddrs);
    student.setAge(studentAge);
    student.setQualification(studentQual);
    student.setPercentage(studentPercent);
    student.setYearPassed(studentYearPass);
    int status=StudentDAO.updateStudent(student);
    if(status>0){
        response.sendRedirect("viewStudents");
    }else{
        out.println("Sorry! unable to update record");
    }
    out.close();
}
}
```

ViewStudents.java

```
package com.srk.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.srk.dao.StudentDAO;
import vo.Student;
@WebServlet("/viewStudents")
public class ViewStudents extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<a href='index.jsp'>Register Student</a>");
        out.println("<h1>Students List</h1>");
        List<Student> list=StudentDAO.getAllStudents();
        out.print("<table border='1' width='100%'>");
        out.print("<tr><th>Student
ID</th><th>StudentName</th><th>Student
Addr</th><th>Student Age</th><th>Student
Qualification</th><th>Student Percentage</th><th>Student
Year Passed</th><th>Edit</th><th>Delete</th></tr>");
    }
}
```

```
for(Student student : list){
    out.print("<tr><td>"+student.getStudentId()+"</td><td>"+s
    tudent.getStudentName()+"</td><td>"+student.getStudentA
    ddr()+"</td><td>"+student.getAge()+"</td><td>"+student.
    getQualification()+"</td><td>"+student.getPercentage()+"</
    td><td>"+student.getYearPassed()+"</td><td><a
    href='editStudent?stdId="+student.getStudentId()+">edit</a
    ></td><td><a
    href='deleteStudent?stdId="+student.getStudentId()+">delet
    e</a></td></tr>");

    }

    out.print("</table>");

    out.close();

}

}
```

SCREENSHOTS

Student registration form

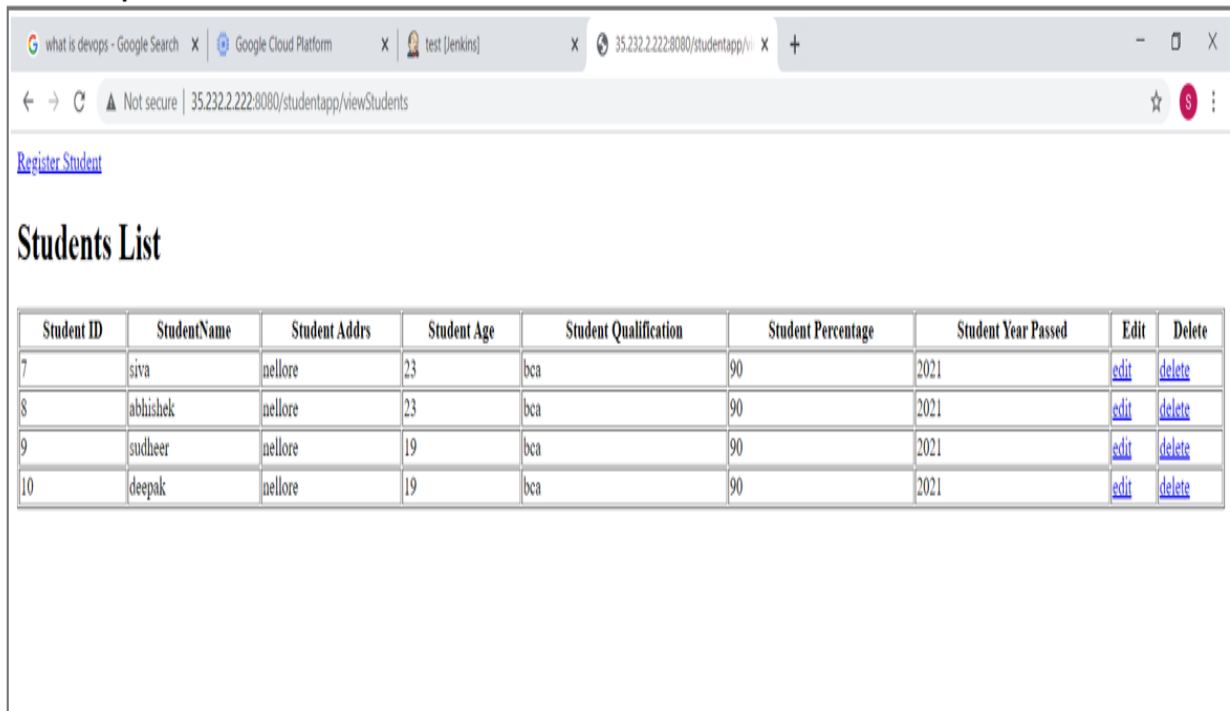
what is devops - Google Search x Google Cloud Platform x test [Jenkins] x User Data x

← → ↻ ⚠ Not secure | 35.232.2.222:8080/studentapp/

Student Registration Form

Student Full Name	<input type="text"/>
Student Permanent Address	<input type="text"/>
Student Age	<input type="text"/>
Student Qualification	<input type="text"/>
Student Percentage	<input type="text"/>
Year Passed	<input type="text"/>
<input type="button" value="register"/>	

output



[Register Student](#)

Students List

Student ID	StudentName	Student Addr	Student Age	Student Qualification	Student Percentage	Student Year Passed	Edit	Delete
7	siva	nellore	23	bca	90	2021	edit	delete
8	abhishek	nellore	23	bca	90	2021	edit	delete
9	sudheer	nellore	19	bca	90	2021	edit	delete
10	deepak	nellore	19	bca	90	2021	edit	delete

CONCLUSION

- DevOps is helping business in a tremendous way.
- It is bridging the gap between developers and operation team and resist change which create a smooth path for continuous development and continuous integration.
- The point is to start somewhere and make small, meaningfull changes. Measure the results and keep iterating. Take advantage of small tools and workflows that devs and ops are using.

BIBLIOGRAPHY

- Javatpoint.com
- Tutorialspoint.com
- Apache.org