

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed

* **Git works on a three-tier model as followes**



* **Working Directory**

This is a local working copy of the project’s latest code.

* **Index (Staging area)**

An Index is the snapshot of your next commit.

It is where the code moves to once you stage or add (git add) the code in the Working directory. A Staging area is like a cache memory and acts as a middle layer between the working directory (where the code is developed) and the local repository (where the code resides).

This layer gives a quick preview of the project snapshot that you are about to commit. You can still edit (add, modify, or delete) your code in the Index.

* **Commit**

A commit is the latest snapshot (state) of a project.



Every commit has a unique commit ID. All commit logs are stored in the local repository.

A commit is a git object that stores the following attributes: commit ID, author name, authored date, and a commit message.

**git hash or the ‘SHA-1’**

The unique commit ID is called a ‘git hash’ or ‘SHA-1’. Every filesystem change (add, delete, edit, move, copy, rename, file permissions, etc.) is treated as a file and its contents are converted into a unique SHA-1 code.

Here is a sample commit ID: **8db083e7df7c9241e640b66c89c6f02649ac885a**

They are often referred by the first 7 unique digits, such as **8db083e**

You never ever have to remember the entire hash ID. git has a beautiful way of handling these commit (hash) IDs using references such as branches and tags.

**Git Checkout**

This command switches over to the specified branch and displays the current project state as it is in the branch.

**Git Clone**

A clone is a working copy of a remote repository.

The ‘git clone’ command downloads the remote repository and creates a working directory on your local machine.

In addition, this command also stores a remote handler or pointer reference from the local repository to the remote repository.



The screenshot shows a local repository — "learn\_branching" — that tracks a remote repository with the URL "<https://github.com/divyabhushan/learn_branching.git>", and "origin" is the name for the remote handler to the remote repository.

**Here are the Git commands**

**gitconfig**

Usage: gitconfig –global user.name “[name]”

Usage: gitconfig –global user.email “[email address]”

This command sets the author name and email address respectively to be used with your commits.



**gitinit**

Usage: gitinit [repository name]

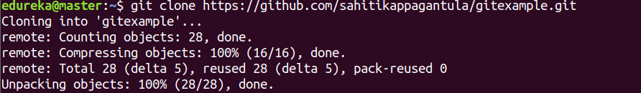
This command is used to start a new repository.



**git clone**

Usage: git clone [url]

This command is used to obtain a repository from an existing URL.



**git add**

Usage: git add [file]

This command adds a file to the staging area.



**git commit**

Usage: git commit -m “[ Type in the commit message]”

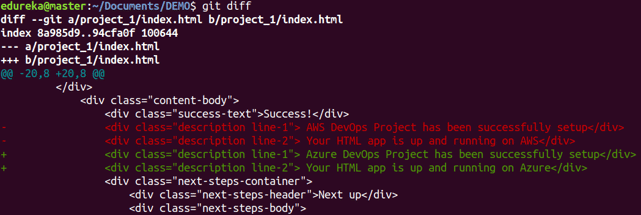
This command records or snapshots the file permanently in the version history.



**git diff**

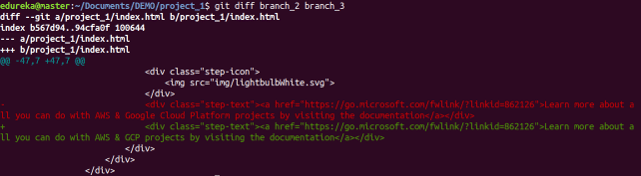
Usage: git diff

This command shows the file differences which are not yet staged.



Usage: git diff [first branch] [second branch]

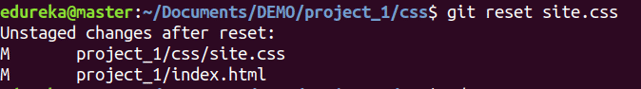
This command shows the differences between the two branches mentioned.



**git reset**

Usage: git reset [file]

This command unstages the file, but it preserves the file contents.



Usage: git reset –hard [commit]

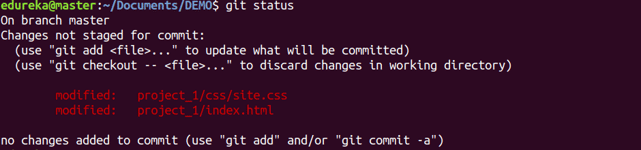
This command discards all history and goes back to the specified commit.



**git status**

Usage: git status

This command lists all the files that have to be committed.



**gitrm**

Usage: gitrm [file]

This command deletes the file from your working directory and stages the deletion.



**git log**

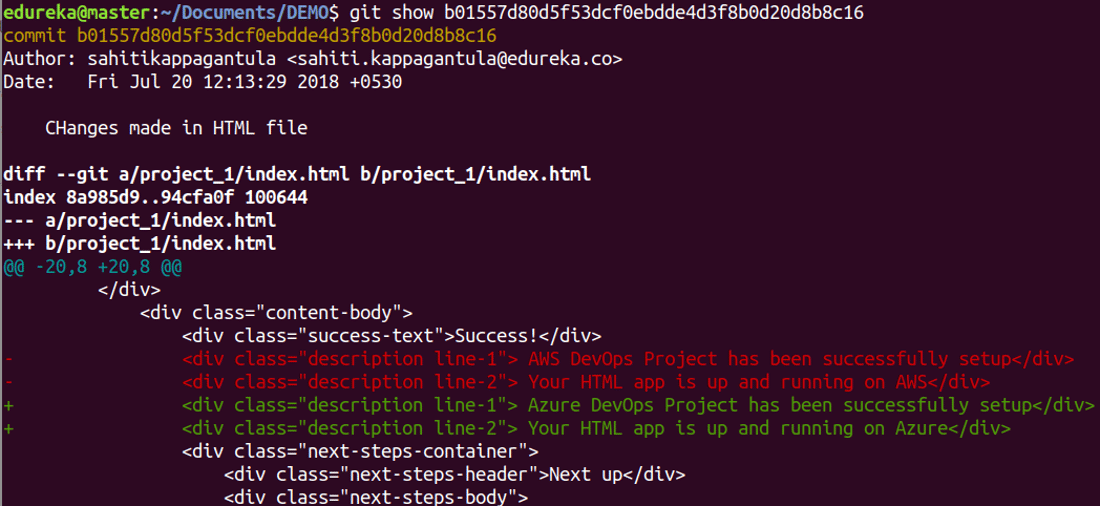
Usage: git log

This command is used to list the version history for the current branch  


**git show**

Usage: git show [commit]

This command shows the metadata and content changes of the specified commit.



**git branch**

Usage: git branch

This command lists all the local branches in the current repository.



**git merge**

Usage: git merge [branch name]

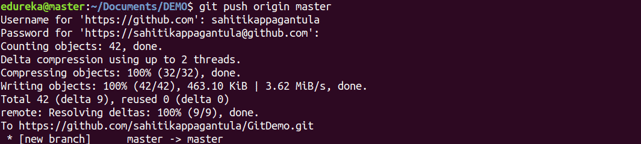
This command merges the specified branch’s history into the current branch.



**git push**

Usage: git push [variable name] master

This command sends the committed changes of master branch to your remote repository.



**git pull**

Usage: git pull [Repository Link]

This command fetches and merges changes on the remote server to your working directory.



* **git branching strategy**

1) Main Branch

2) Supporting Branch

**1) Main branch**

These branches have an infinite lifetime i.e **Master and Develop Branch**

* **Master branch**

This branch always points to latest stable release (production-ready state).

It's equivalent to release candidate in release process.

* **Develop branch**

This branch always points to latest development code.

It's equivalent to alpha state in release process.

**2) Supporting branch**

These branches have limited life-time with specific purpose and it will be removed eventually once it's merged with corresponding branches.

* **Feature branch**

This branch is used to develop new features / enhance existing feature.

This branch should branch off from: develop.

This branch should merge back into: develop.

Note: The feature branch should be merged into develop branch only after peer review, code review and QA Sign-off.

* **Bug branch**

This branch is used to fix the bugs in existing system.

This branch should branch off from: develop.

This branch should merge back into: develop.

Note: The feature branch should be merged into develop branch only after peer review, code review and QA Sign-off.

* **Release branch**

It's equivalent to beta state in release process.

This branch is used for UAT.

Issues will be fixed in this branch.

This brach should branch off from: develop

This branch should merge back into: master and develop

The master branch merge commit will be tagged with the new version number.

* **Hotfix branch**

This branch is used to fix P1 issues.

This branch should branch off from: master

This branch should merge back into: master and develop



Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. ... In short terms we can tell maven is a tool that can be used for building and managing any Java-based project.

How we can generate new project by using mavan

**mvnarchetype:generatemvn = maven project archetype = plugin**

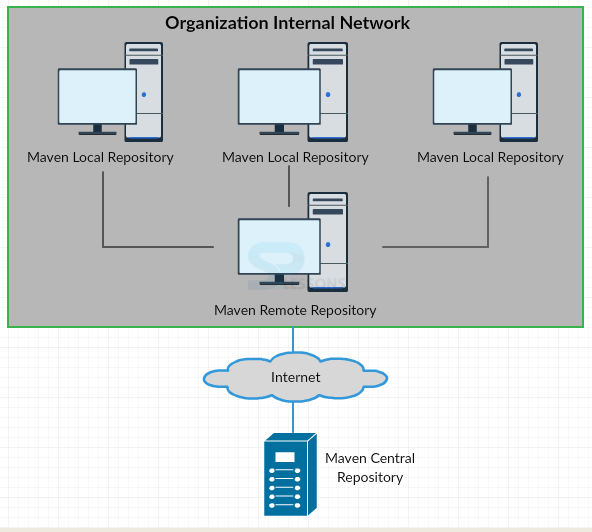
**generate = gole**

when we hit this cmd (**mvnarchetype:generate**) mvn will get the plugins like this

firstmvn check in its local Repository if its not there it go to

Remote Repository from remote it will get all plugins and store in .m2 in local repo.

Central repository its third party tool not releted to mvn



**Maven Lifecycle**

* **validate** - validate the project is correct and all necessary information is available
* **compile** - compile the source code of the project
* **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* **package** - take the compiled code and package it in its distributable format, such as a JAR.
* **install** - install the package into the local repository, for use as a dependency in other projects locally
* **deploy**- done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

**Maven Overview - Core Concepts**

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



**Maven Directory Structure**

- src

- main

- java

- resources

- webapp

- test

- java

- resources

- target

**src directory** is the root directory of your source code and test code.

**main directory** is the root directory for source code related to the application itself (not test code).

**test directory** contains the test source code.

**java directories** under main and test contains the Java code for the application itself (under main) and the Java code for the tests (under test).

**resources directory** contains other resources needed by your project. This could be property files used for internationalization of an application

**webapp directory** contains your Java web application, if your project is a web application. The webapp directory will then be the root directory of the web application. Thus the webapp directory contains the WEB-INF directory etc.

**target directory** is created by Maven. It contains all the compiled classes, JAR files etc. produced by Maven. When executing the clean build phase, it is the target directory which is cleaned.

**Jenkins**

****

**What is Jenkins**

Jenkins is an open source Continuous Integration server capable of orchestrating a chain of actions that help to achieve the Continuous Integration process (and not only) in an automated fashion.

Jenkins is free and is entirely written in Java. Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

It is a server-based application and requires a web server like Apache Tomcat. The reason Jenkins became so popular is that of its monitoring of repeated tasks which arise during the development of a project. For example, if your team is developing a project, Jenkins will continuously test your project builds and show you the errors in early stages of your development.

By using Jenkins, software companies can accelerate their software development process, as Jenkins can automate build and test at a rapid rate. Jenkins supports the complete development lifecycle of software from building, testing, documenting the software, deploying and other stages of a software development lifecycle.

**why we use Jenkins**

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build



Jenkins is an open source automation server that offers an easy way to set up a continuous integration and continuous delivery (CI/CD) pipeline.

Continuous integration (CI) is a DevOps practice in which team members regularly commit their code changes to the version control repository, after which automated builds and tests are run. Continuous delivery (CD) is a series of practices where code changes are automatically built, tested and deployed to production.

**How to Installing Jenkins in ubuntu**

**Install Java.**

Since Jenkins is a Java application, the first step is to install Java. Update the package index and install the Java 8 OpenJDK package with the following commands:

**sudo apt update**

**sudo apt install openjdk-8-jdk**

The current version of Jenkins does not support Java 10 (and Java 11) yet. If you have multiple versions of Java installed on your machine make sure Java 8 is the default Java version.

Add the Jenkins Debian repository.

Import the GPG keys of the Jenkins repository using the following wget command:

**wget -q -O -** [**https://pkg.jenkins.io/debian/jenkins.io.key**](https://pkg.jenkins.io/debian/jenkins.io.key) **| sudo apt-key add -**

The command above should output OK which means that the key has been successfully imported and packages from this repository will be considered trusted.

Next, add the Jenkins repository to the system with:

**sudosh -c 'echo deb <**[**http://pkg.jenkins.io/debian-stable binary/**](http://pkg.jenkins.io/debian-stable%20binary/)**>> /etc/apt/sources.list.d/jenkins.list'**

**Install Jenkins**.

Once the Jenkins repository is enabled, update the apt package list and install the latest version of Jenkins by typing:

**sudo apt update**

**sudo apt install jenkins**

Jenkins service will automatically start after the installation process is complete. You can verify it by printing the service status:

**systemctl status jenkins**

You should see something similar to this:

●jenkins.service - LSB: Start Jenkins at boot time

Loaded: loaded (/etc/init.d/jenkins; generated)

Active: active (exited) since Wed 2018-08-22 13:03:08 PDT; 2min 16s ago

Docs: man:systemd-sysv-generator(8)

Tasks: 0 (limit: 2319)

CGroup: /system.slice/jenkins.service

Setting Up Jenkins

To set up your new Jenkins installation, open your browser, type your domain or IP address followed by port 8080, <http://your_ip_or_domain:8080> and screen similar to the following will be displayed**:**

****

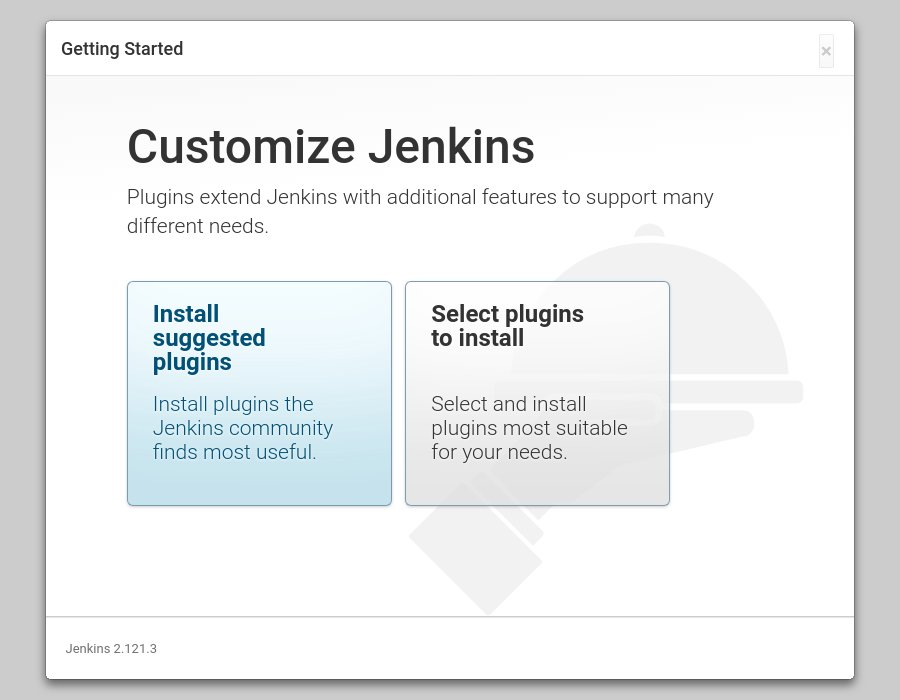
During the installation, the Jenkins installer creates an initial 32-character long alphanumeric password. Use the following command to print the password on your terminal

**sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

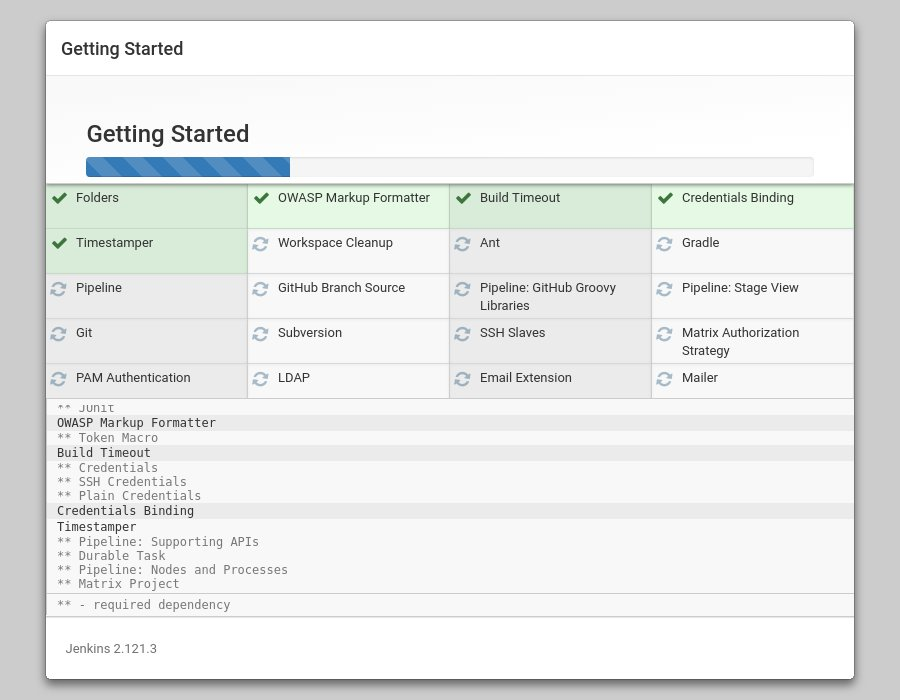
output

**2115173b548f4e99a203ee99a8732a32**

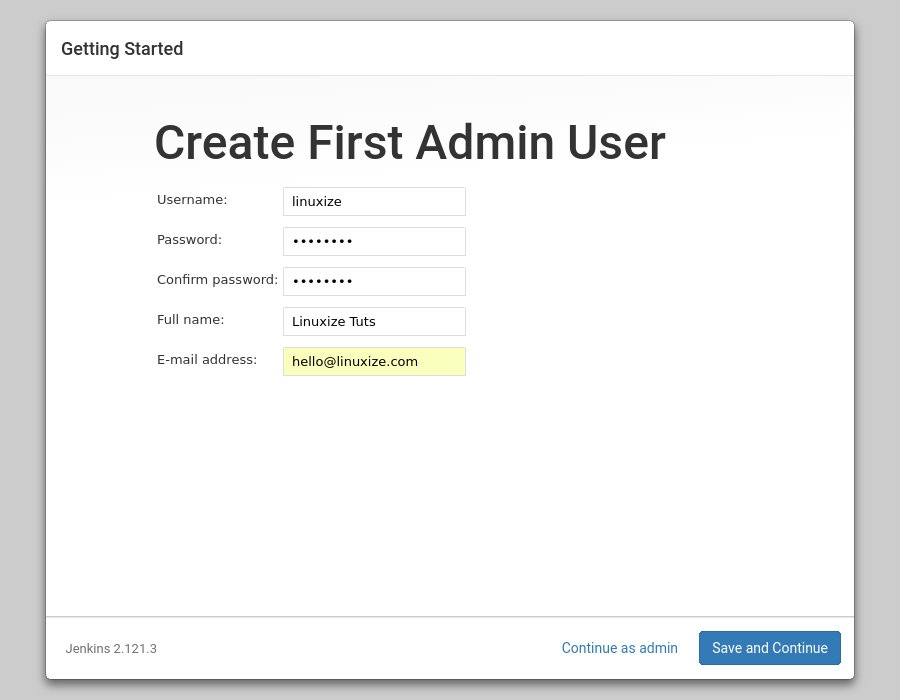
Copy the password from your terminal, paste it into the Administrator password field and click **Continue.**

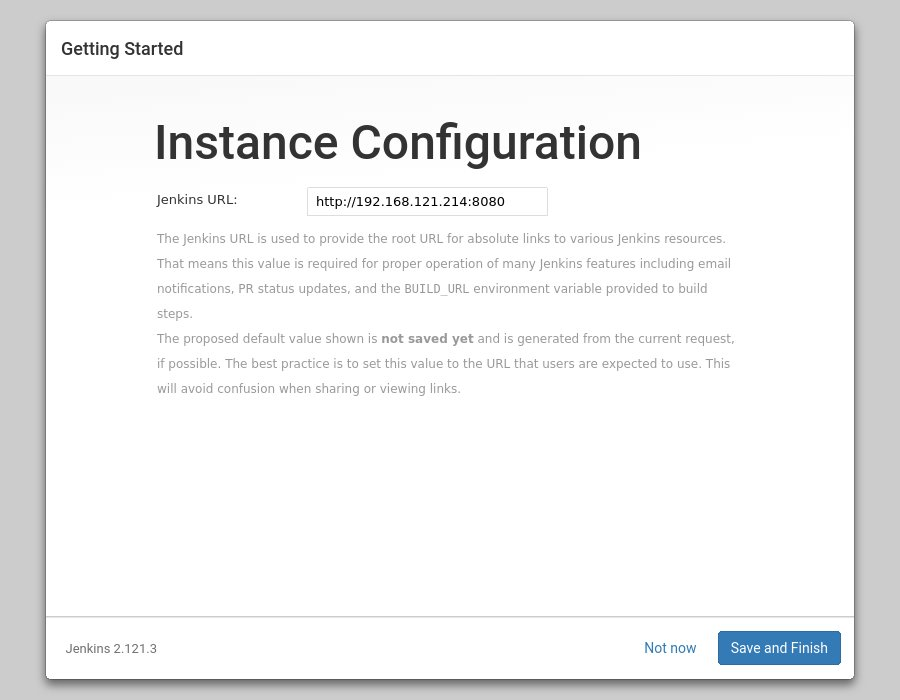
****

On the next screen, the setup wizard will ask you whether you want to install suggested plugins or you want to select specific plugins. Click on the Install suggested plugins box, and the installation process will start immediately.

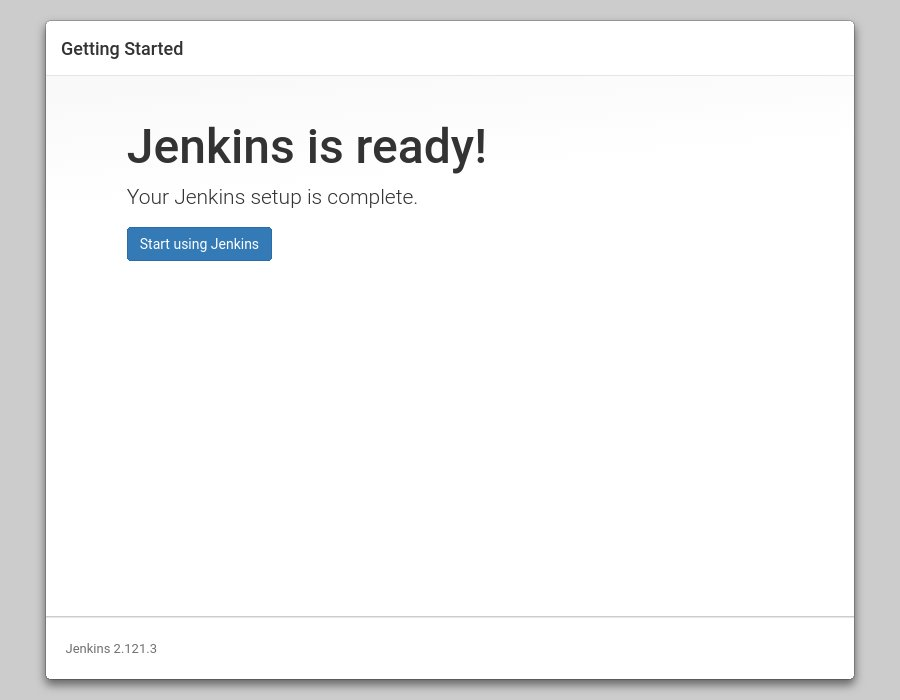


Once the plugins are installed, you will be prompted to set up the first admin user. Fill out all required information and **click Save and Continue.**

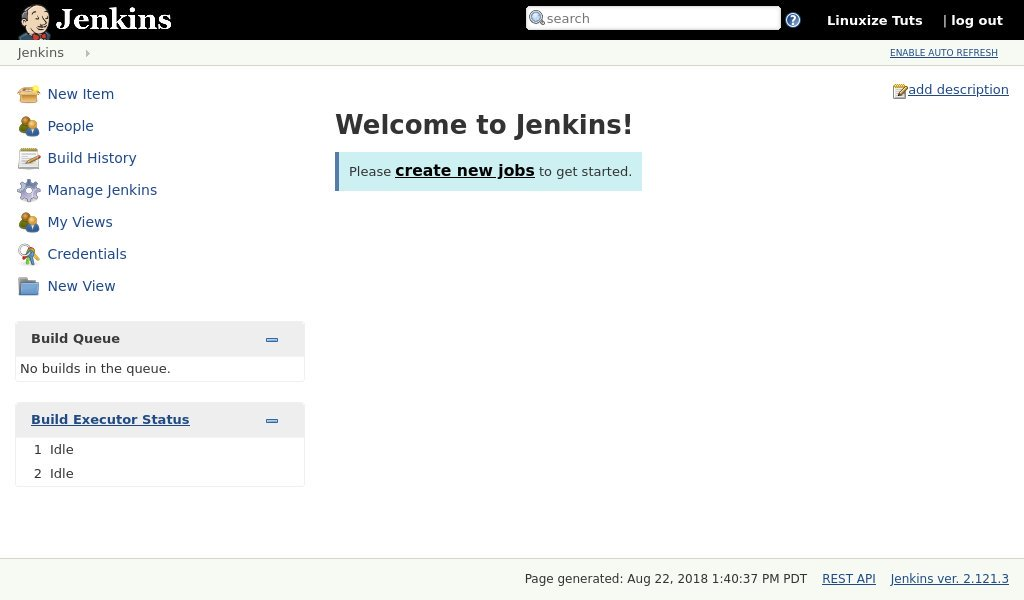
**  
The next page will ask you to set the URL for your Jenkins instance. The field will be populated with an automatically generated URL.**

****

**Confirm the URL by clicking on the Save and Finish button and the setup process will be com**

****

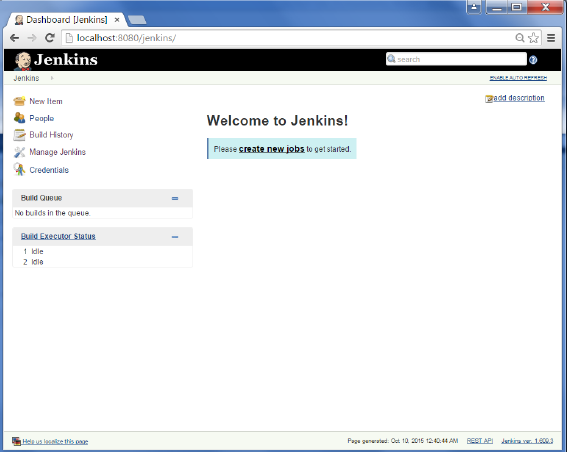
**pleted.**Click on the **Start using Jenkins** button and you will be redirected to the Jenkins dashboard logged in as the admin user you have created in one of the previous steps.



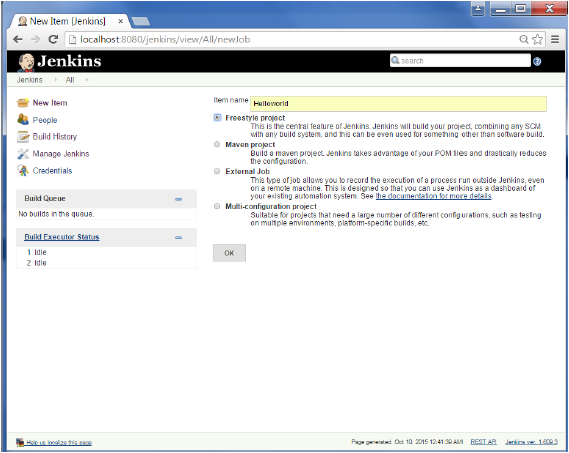
At this point, you’ve successfully installed Jenkins on your system.\

**How to create Job in Jenkins**

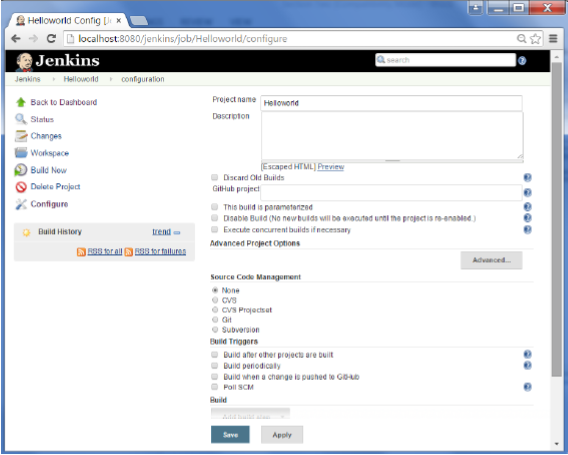
**Step 1 −**Go to the Jenkins dashboard and Click on **New Item**

****

**Step2 −** Inthenextscreen, entertheItemname, inthiscasewehavenameditHelloworld. **Choosethe‘Freestyleprojectoption’**

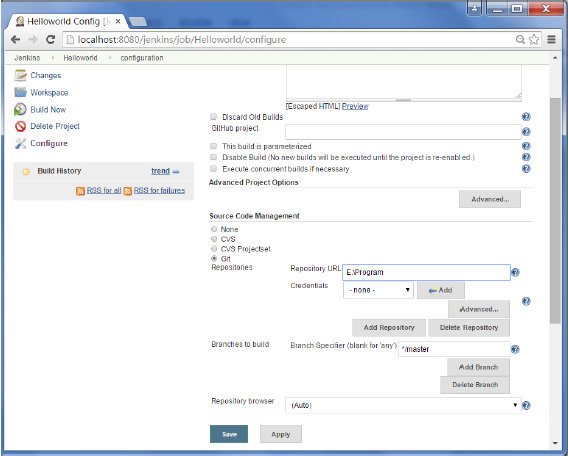
****

**Step3 − The following screen will come up in which you can specify the details of the job**

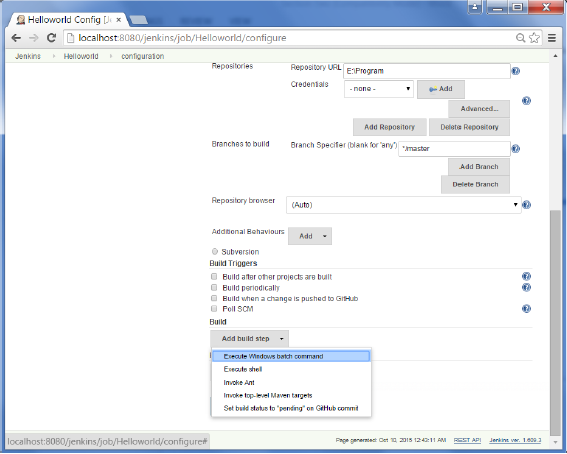
****

**Step 4** − We need to specify the location of files which need to be built. In this example, we will assume that a local gitrepository(E:\Program) has been setup which contains a ‘HelloWorld.java’ file. Hence scroll down and click on the Git option and enter the URL of the local git repository.

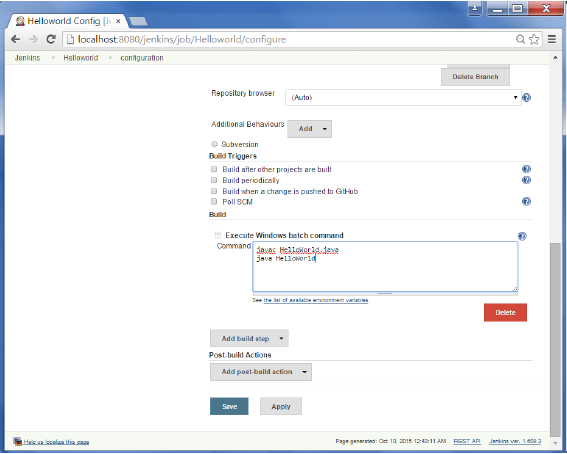
**Note −** If you repository if hosted on Github, you can also enter the url of that repository here. In addition to this, you would need to click on the Add button for the credentials to add a user name and password to the github repository so that the code can be picked up from the remote repository



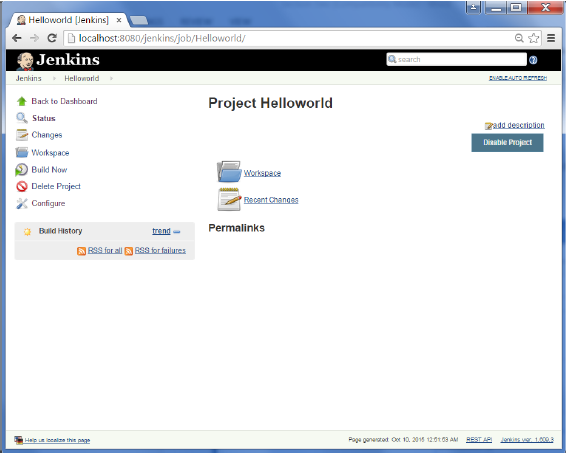
**Step** 5 − Now go to the Build section and click on Add build step → Execute Windows batch command



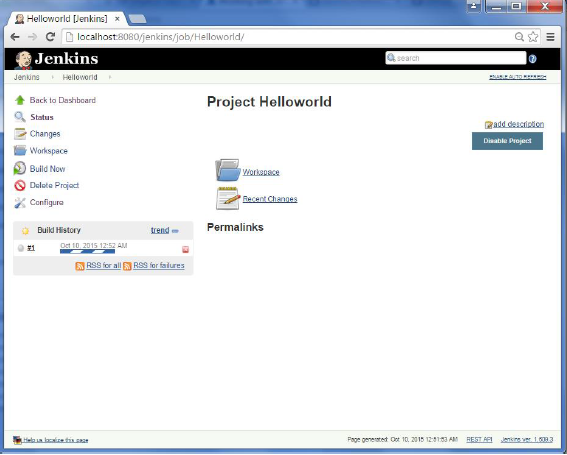
**Step 6** − In the command window, enter the following commands and then click on the Save button



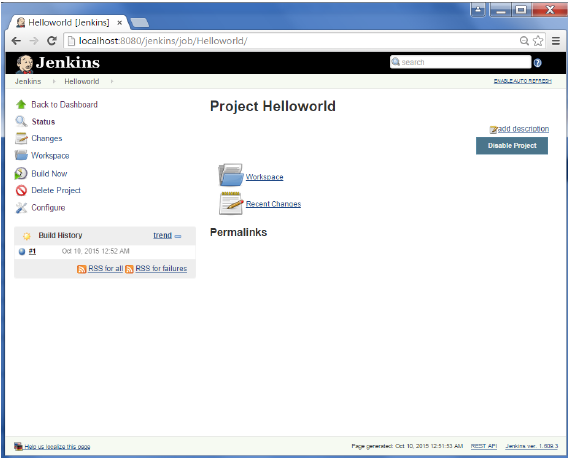
**Step** 7 − Once saved, you can click on the Build Now option to see if you have successfully defined the job.



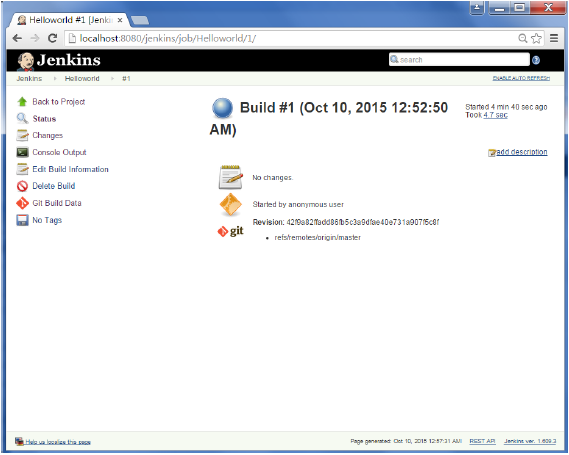
**Step 8** − Once the build is scheduled, it will run. The following Build history section shows that a build is in progress.



**Step 9** − Once the build is completed, a status of the build will show if the build was successful or not. In our case, the following build has been executed successfully. Click on the #1 in the Build history to bring up the details of the build



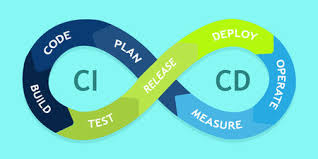
**Step 10** − Click on the Console Output link to see the details of the build





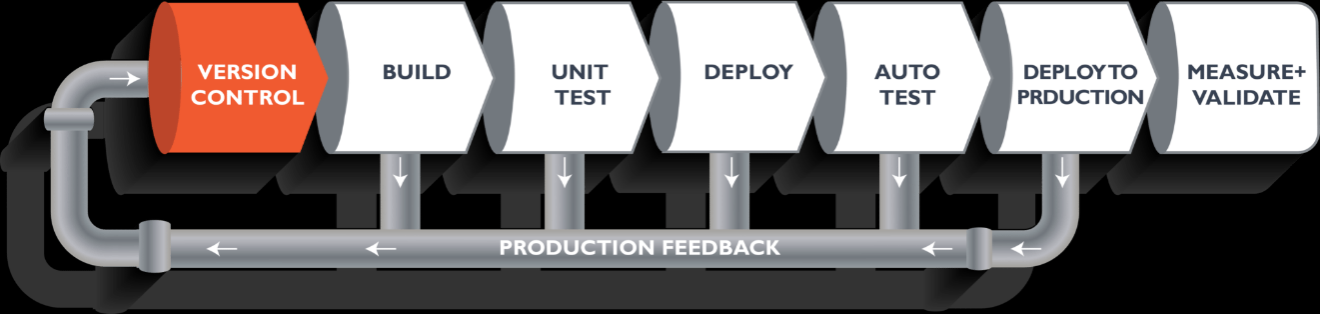
Apart from the steps shown above there are just so many ways to create a build job, the options available are many, which what makes Jenkins such a fantastic continuous deployment tool.

**CI/CD Pipeline**

****

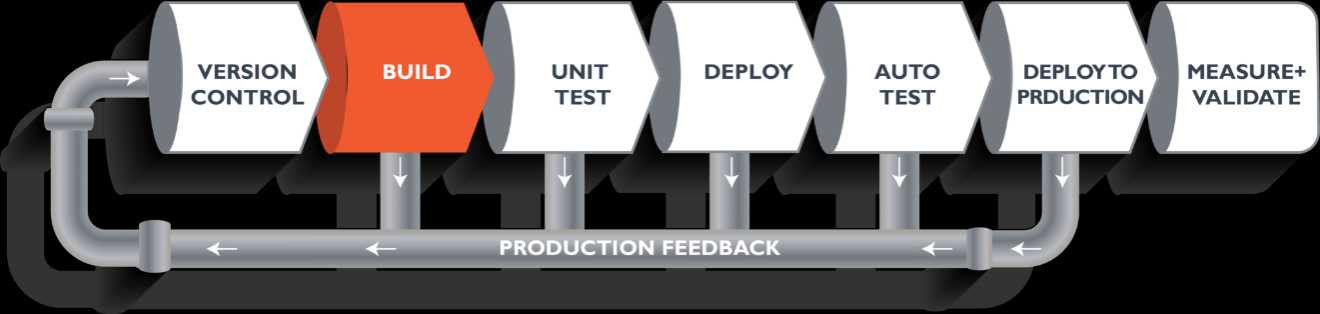
CI stands for Continuous Integration and CD stands for Continuous Delivery/Continuous Deployment. You can think of it as a process similar to a software development lifecycle.

Let us see how it works.

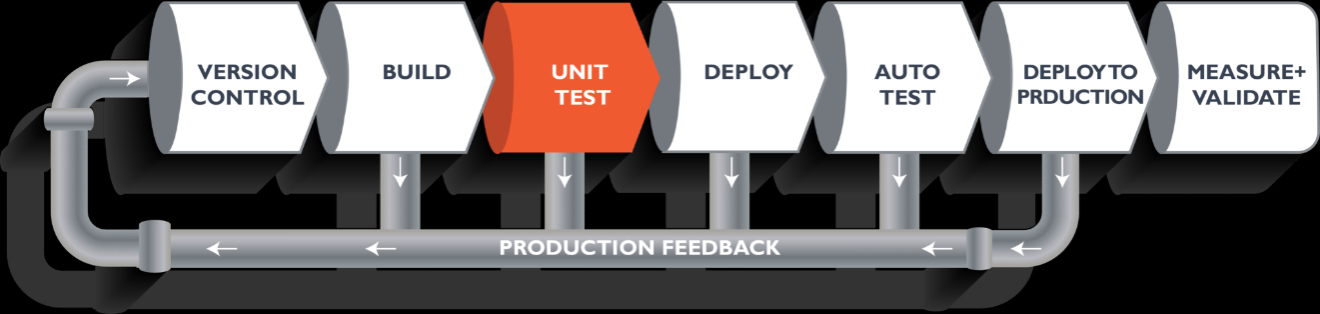


The above pipeline is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production.

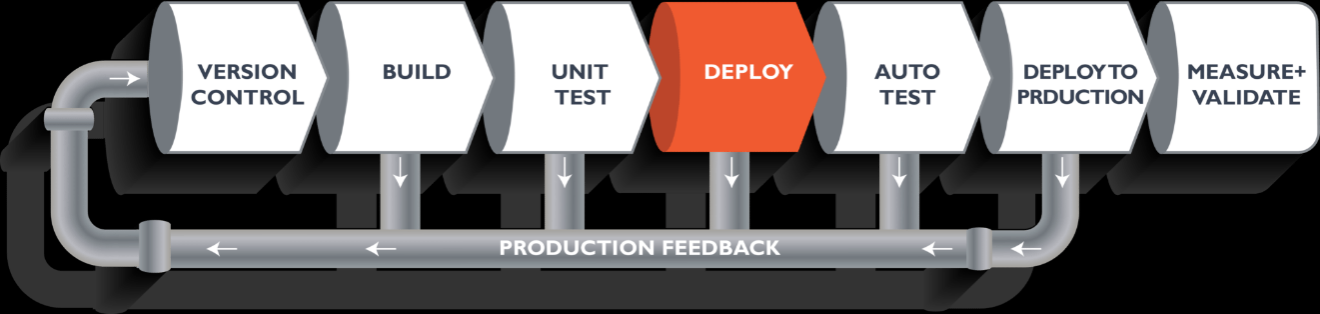
Let's take a scenario of a CI/CD Pipeline. Imagine you're going to build a web application which is going to be deployed on live web servers. You will have a set of developers responsible for writing the code, who will further go on and build the web application. Now, when this code is committed into a version control system (such as git, svn) by the team of developers. Next, it goes through the build phase, which is the first phase of the pipeline, where developers put in their code and then again the code goes to the version control system with a proper version tag.



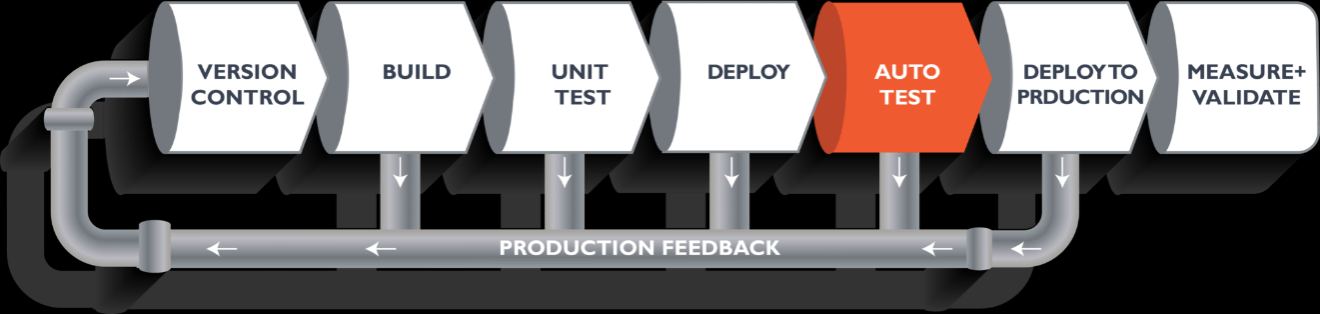
Suppose we have Java code and it needs to be compiled before execution. Through the version control phase, it again goes to the build phase, where it is compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it. This whole process is called the build phase.



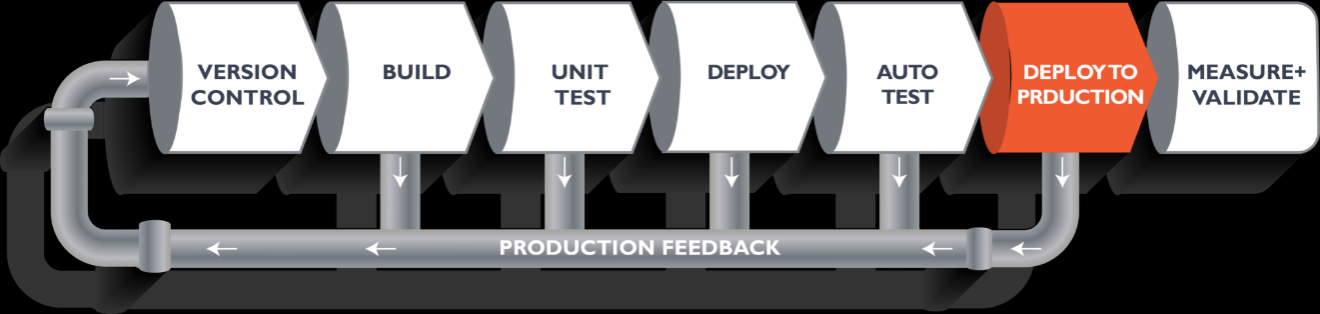
Once the build phase is over, then you move on to the testing phase. In this phase, we have various kinds of testing. One of them is the unit test (where you test the chunk/unit of software or for its sanity test).



When the test is completed, you move on to the deploy phase, where you deploy it into a staging or a test server. Here, you can view the code or you can view the app in a simulator.

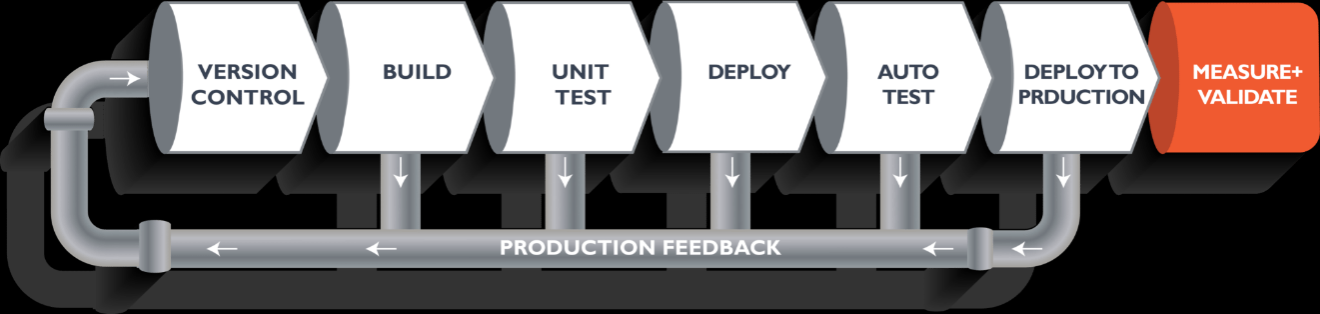


Once the code is deployed successfully, you can run another sanity test. If everything is accepted, then it can be deployed to production.



Meanwhile, in every step, if there is an error, you can shoot an email back to the development team so that they can fix it. Then they will push it into the version control system and it goes back into the pipeline.

Once again, if there is any error reported during testing, the feedback goes to the dev team again, where they fix it and the process reiterates if required.



This lifecycle continues until we get code/a product which can be deployed to the production server where we measure and validate the code.

We now understand the CI/CD Pipeline and its working; now, we will move on to understand what Jenkins is and how we can deploy the demonstrated code using Jenkins and automate the entire process.

**Building a CI/CD Pipeline Using Docker and Jenkins**

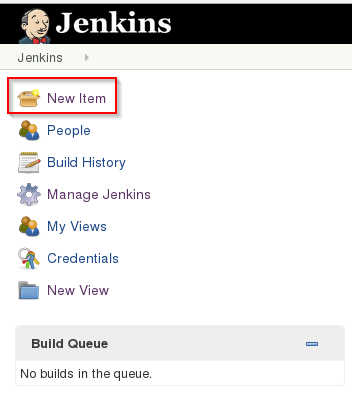
**Step 1**: Open your terminal in your VM. Start Jenkins and Docker using these commands:

systemctl start jenkins

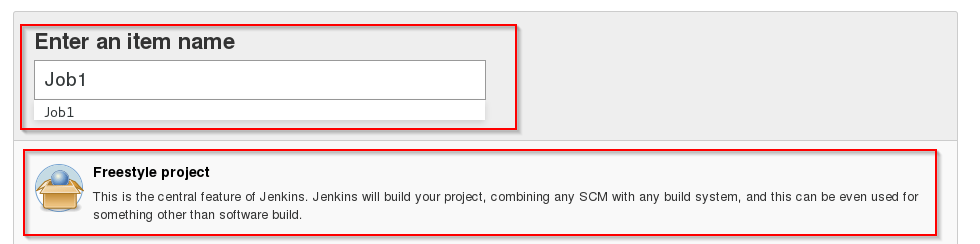
systemctl enable jenkins

systemctl start docker

**Step 2**: Open Jenkins on your specified port. Click on New Item to create a Job.



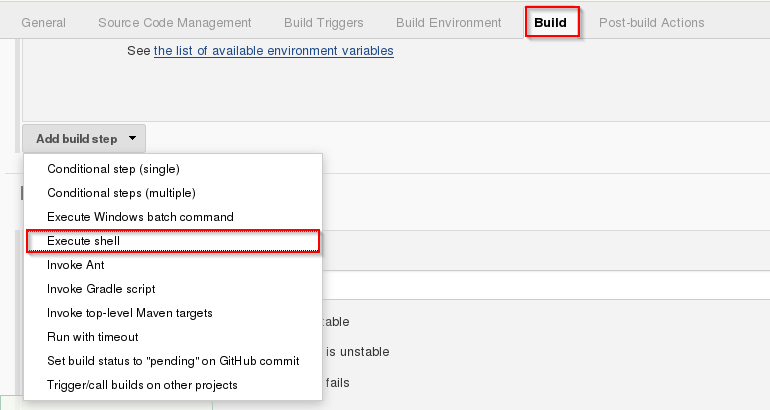
**Step** 3: Select a freestyle project and provide the item name (here I have given Job1) and **click OK.**

****

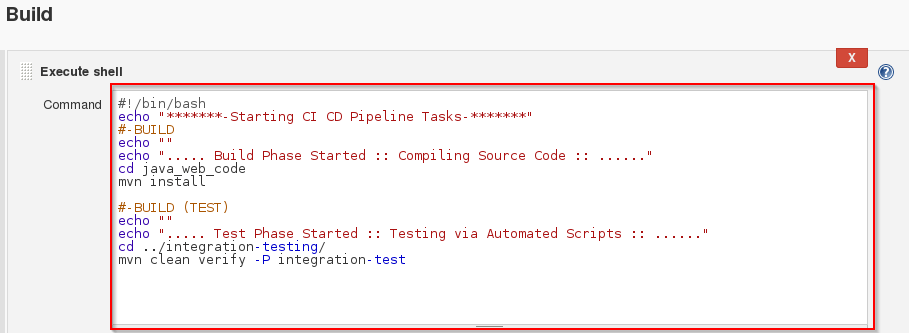
**Step** 4: Select Source Code Management and provide the Git repository. Click on **Apply and Save button**.



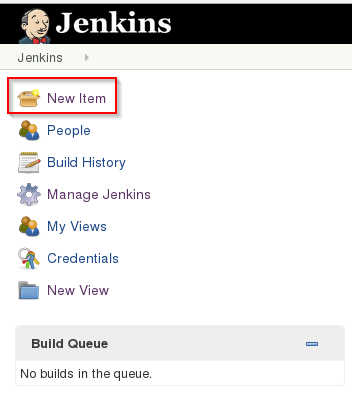
**Step** 5: Then click on **Build->Select Execute Shell**.



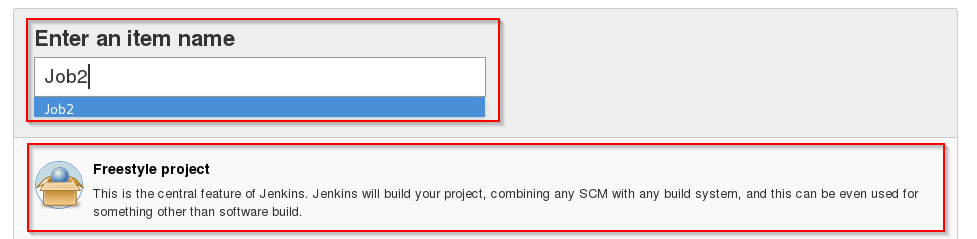
**Step** 6: Provide the shell commands. Here, it will build the archive file to get a war file. After that, it will get the code which is already pulled and then it uses maven to install the package. It simply installs the dependencies and compiles the application.



**Step** 7: Create the new Job by clicking on New Item

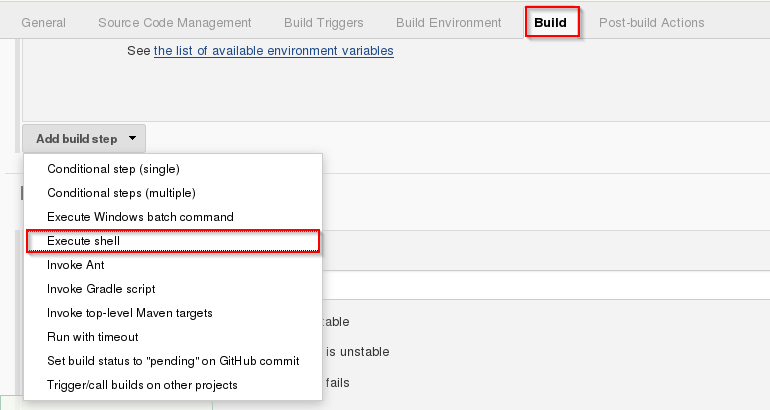


**Step** 8: Select freestyle project and provide the item name (here I have given Job2) and click on OK.

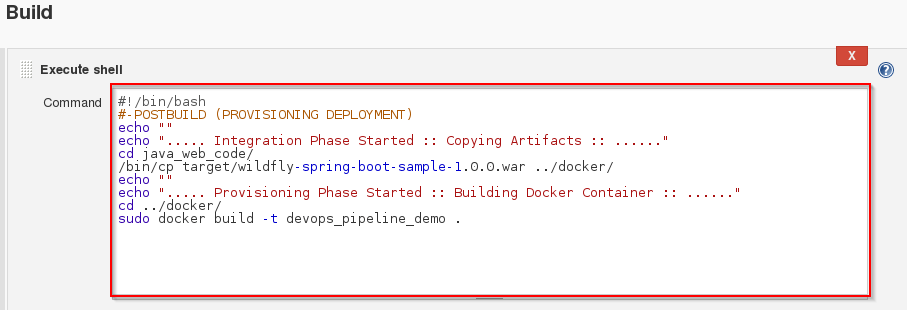
  
**Step** 9: Select Source Code Management and provide the Git repository. Click on Apply and Save button



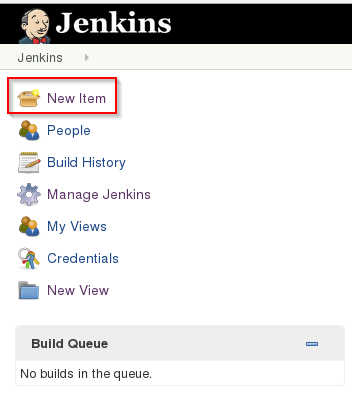
**Step** 10: Then click on Build->Select Execute Shell.



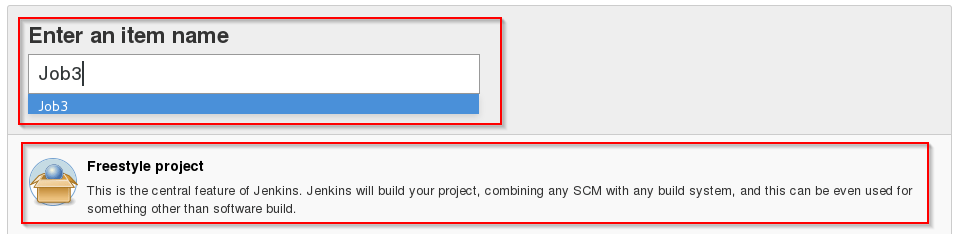
**Step** 11: Provide the shell commands. Here it will start the integration phase and build the Docker Container.



**Step** 12: Create the new Job by clicking on New Item.



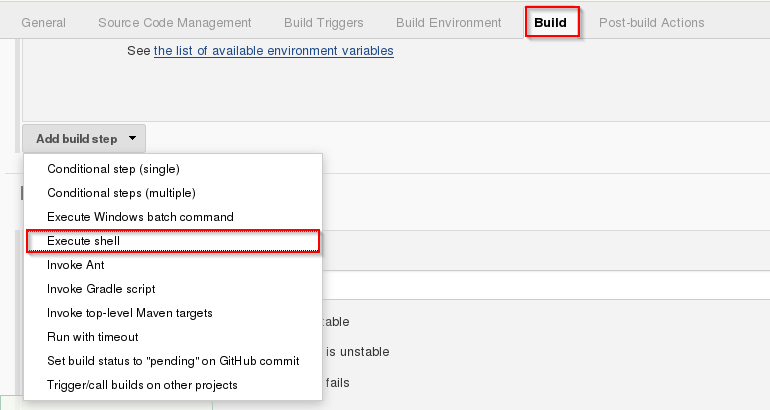
**Step** 13: Select freestyle project and provide the item name (here I have given Job3) and click on OK.



**Step** 14: Select Source Code Management and provide the Git repository. Click on Apply and Save button.

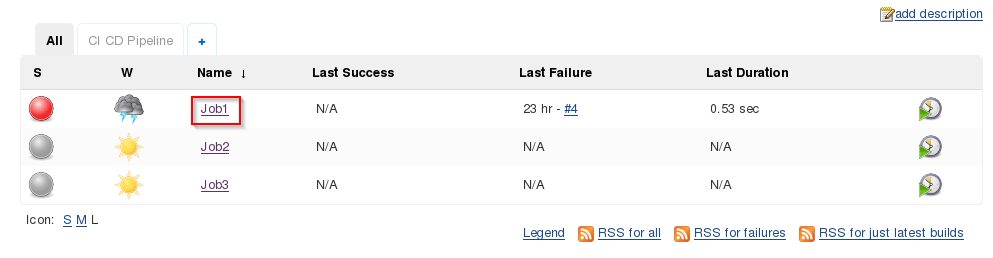


**Step** 15: Then click on Build->Select Execute Shell.

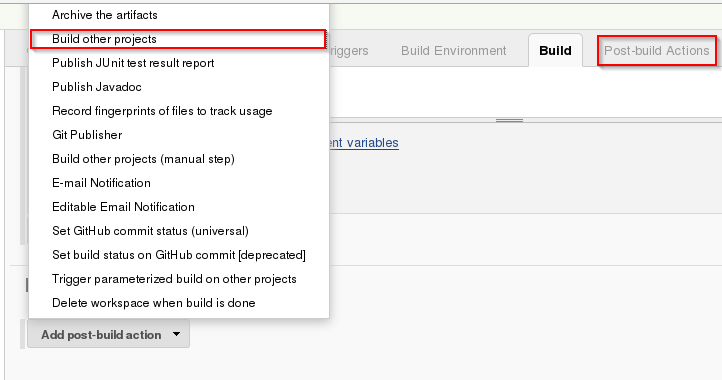


**Step** 16: Provide the shell commands. Here it will check for the Docker Container file and then deploy it on port number 8180. Click on Save button.



**step** 17: Now click on Job1 -> Configure.  
****

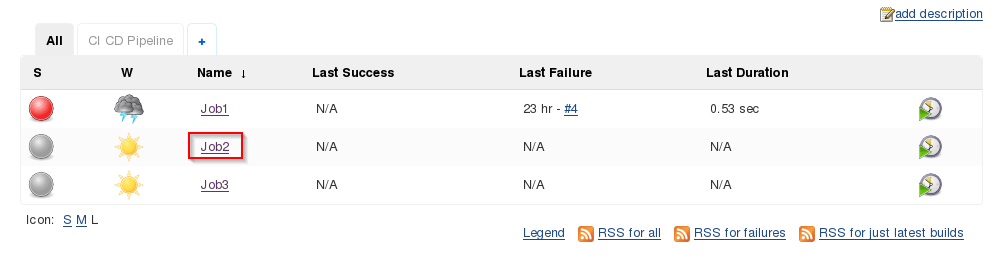
**Step** 18: Click on Post-build Actions -> Build other projects.



**Step** 19: Provide the project name to build after Job1 (here is Job2) and then click on Save.

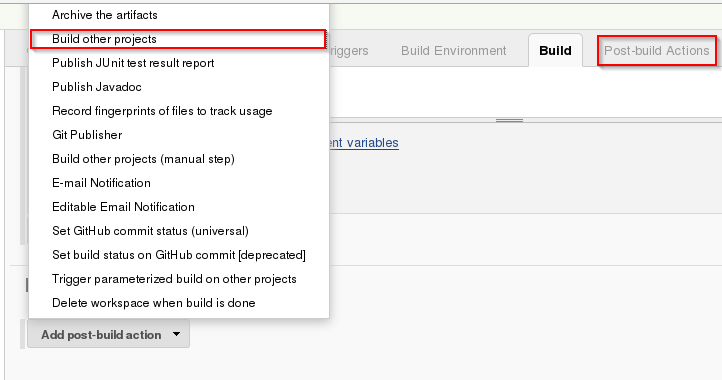


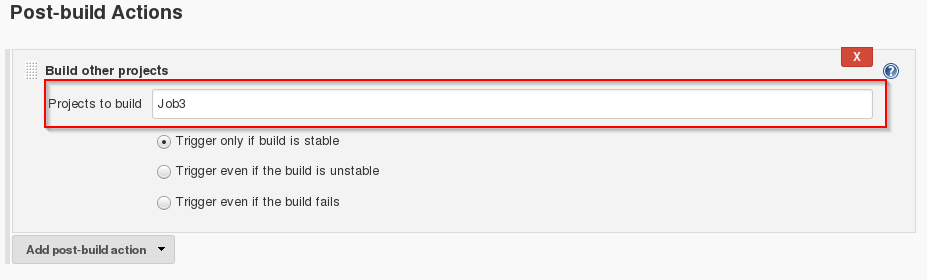
Step 20: Now click on Job2 -> Configure.



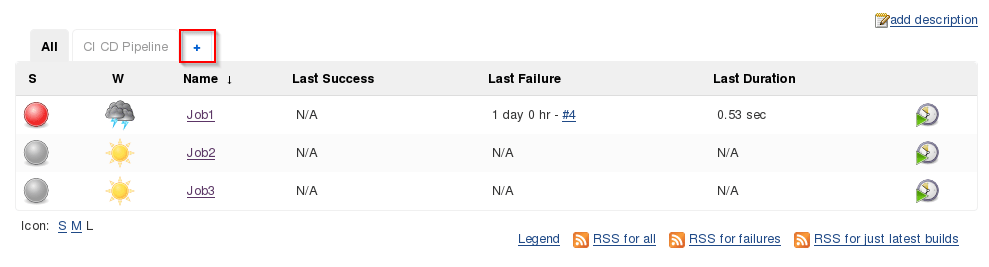


**Step** 21: Click on Post-build Actions -> Build other projects.

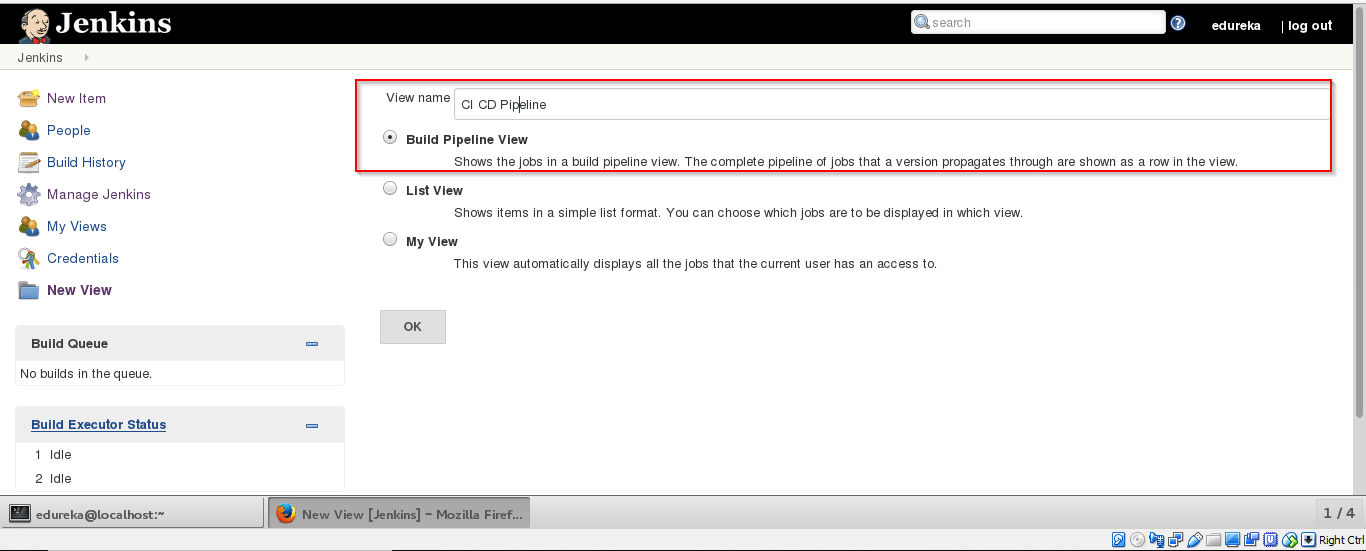
 **Step** 22: Provide the project name to build after Job2 (here is Job3) and then click on Save.



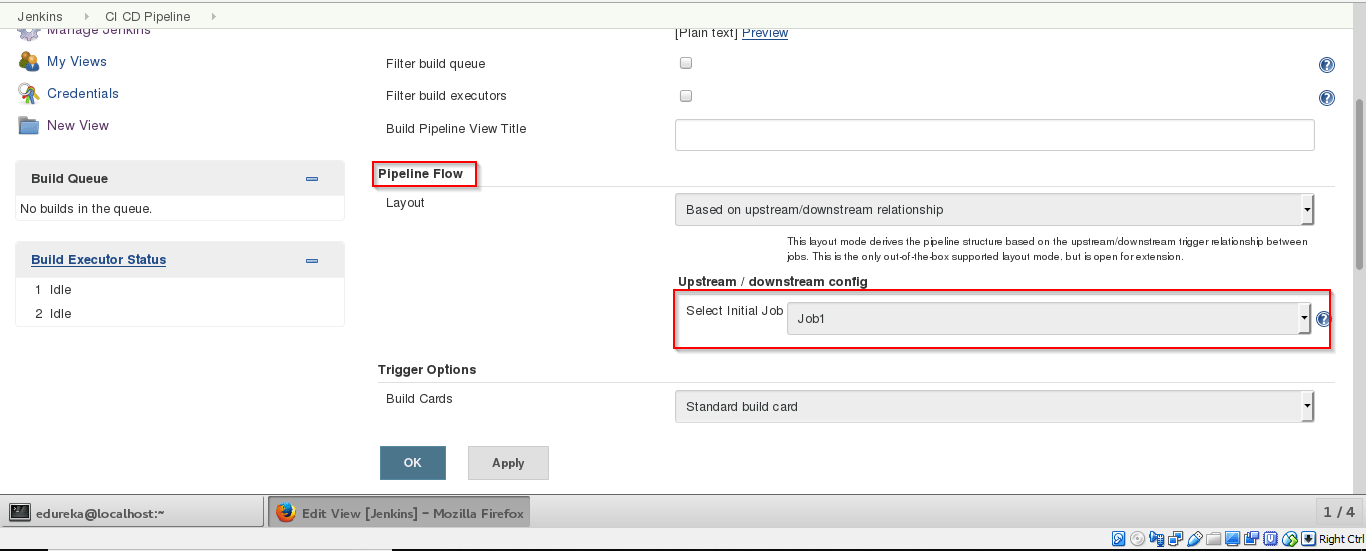
Step 23: Now we will be creating a Pipeline view. Click on the "+" sign.



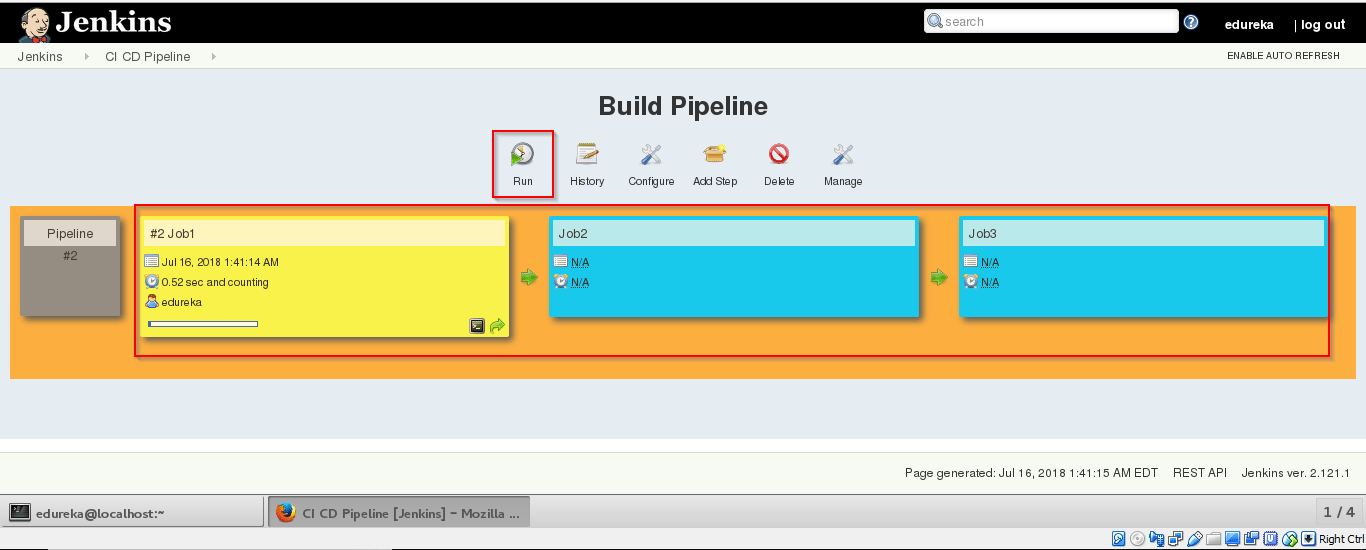
**Step** 24: Select Build Pipeline View and provide the view name (here I have provided CI CD Pipeline).



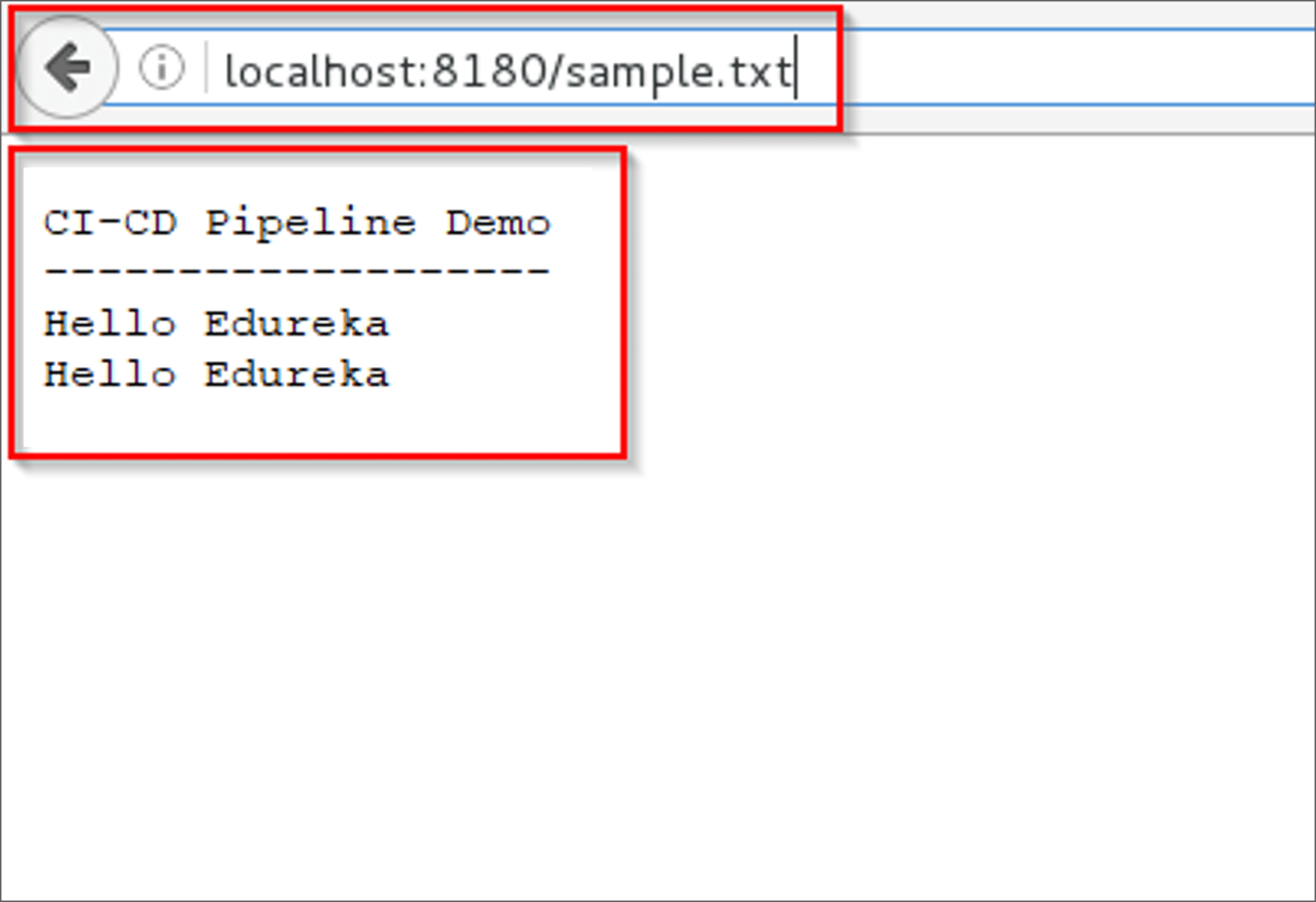
Step 25: Select the initialJob (here I have provided Job1) and click on OK.



**Step** 26: Click on Run button to start the CI/CD process.



**Step** 27: After successful build open localhost:8180/sample.text. It will run the application.



So far, we have learned how to create a CI/CD Pipeline using Docker and Jenkins. The intention of DevOps is to create better-quality software more quickly and with more reliability while inviting greater communication and collaboration between teams.