**GITLAB CI/CD PIPELINE**

**=====================**

**Agenda :-**

**1🡪 Introduction to CI/CD**

**2🡪CI-CD Pipeline**

**3🡪 GItlab CI-CD**

**4🡪Buildings a CI-CD pipeline**

**5🡪Conclusion**

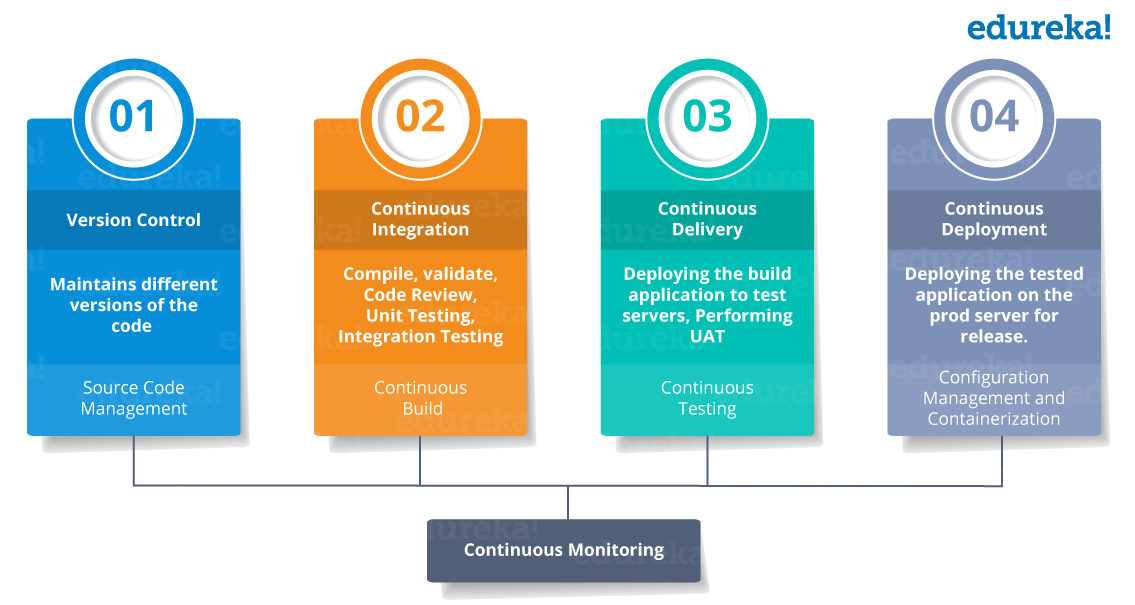
**1🡪 Introduction to CI/CD=**

**🡪** Continuous integration, delivery, and deployment is known as **CI/CD**.

**🡪 C I/CD** essentially involves continuously building, testing and deploying code changes at every small integration, reducing the chance of deploying new code based on budget or failed previous version.

Reduce error in code, Speeds up Coding Process , Integrates Code Seamlessly

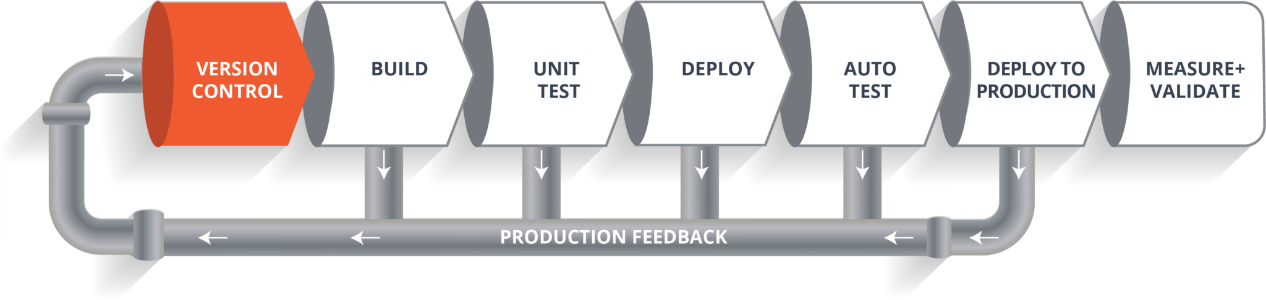
**🡪**There are Four main approaches to this methodology:-



**2🡪CI-CD Pipeline =**

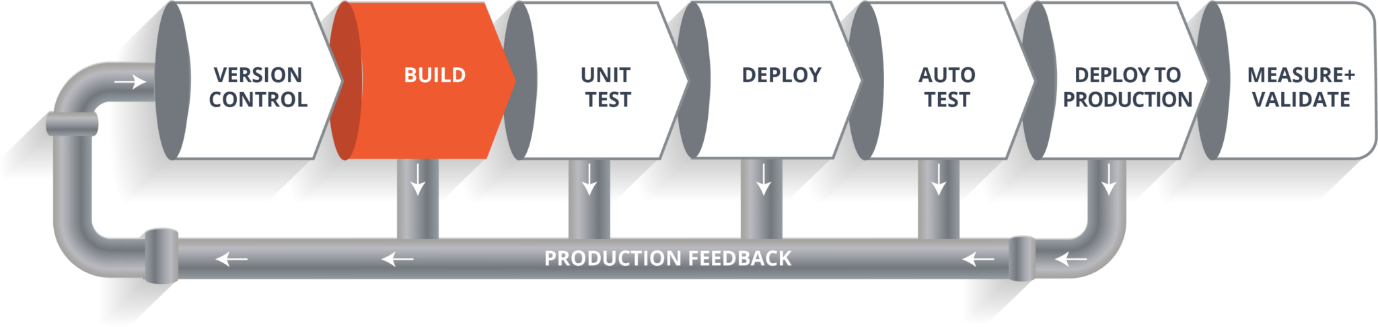
**🡪** CI stands for Continuous Integration and CD stands for Continuous Delivery and Continuous Deployment. You can think of it as a process which is similar to a software development lifecycle.

**🡪**  Now let us see how does it’s work.



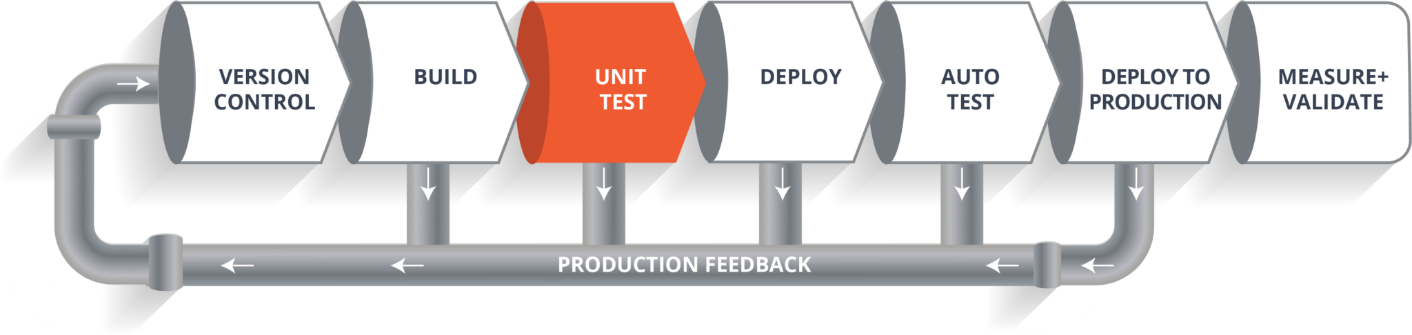
The above pipeline is a logical demonstration of how a software will move along the various phases or stages in this lifecycle, before it is delivered to the customer or before it is live on production.

Let’s take a scenario of 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 who are responsible for writing the code which 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 code goes to the version control system having a proper version tag.

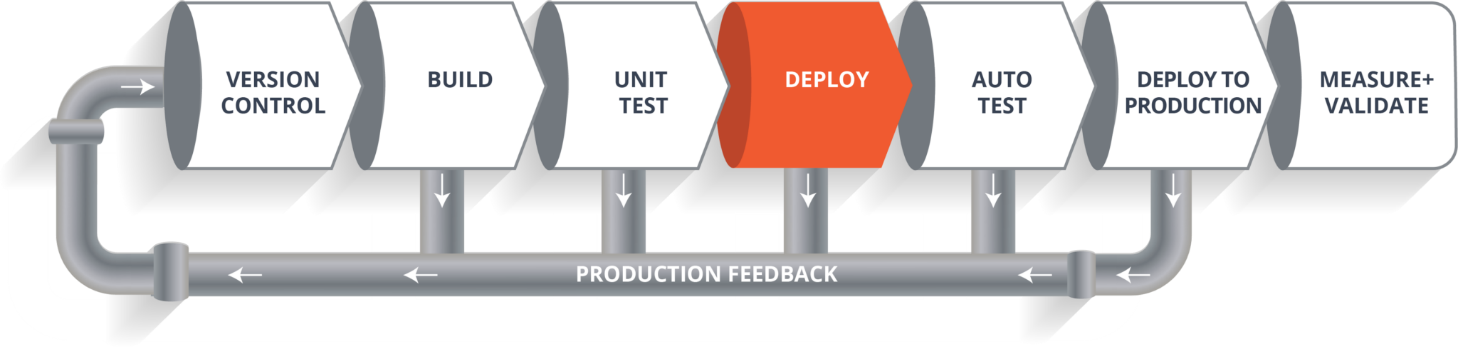


Suppose we have a Java code and it needs to be compiled before execution. So, through the version control phase, it again goes to build phase where it gets compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it.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

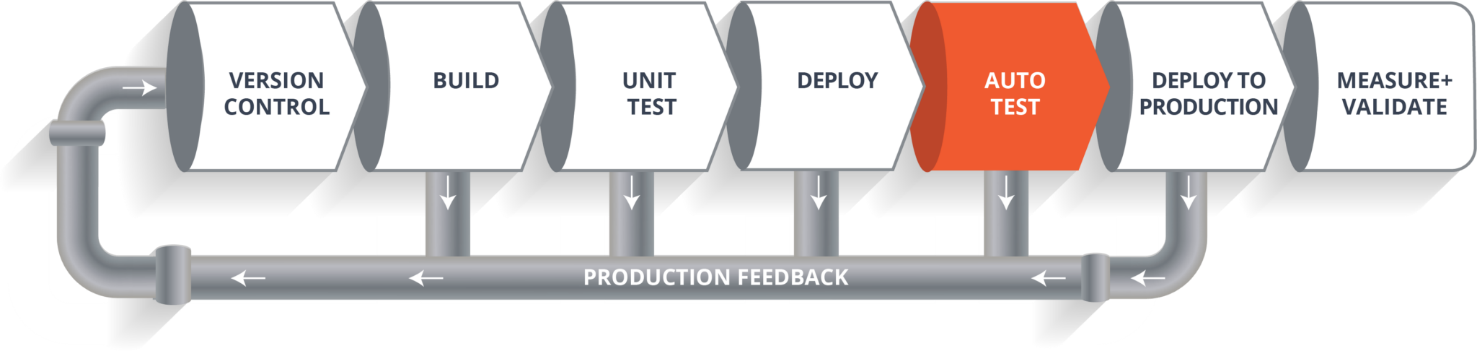
### ****Testing Phase:****

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).

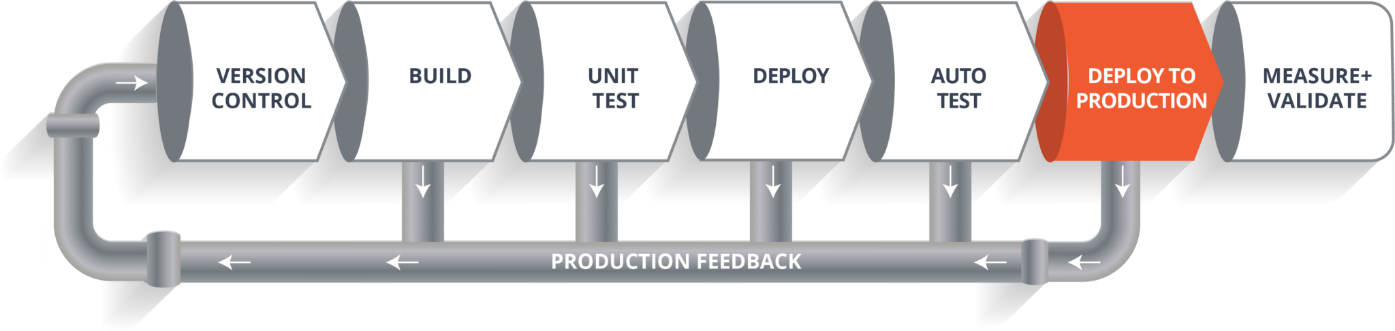
### ****Deploy Phase:****

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.

### ****Auto Test Phase:****

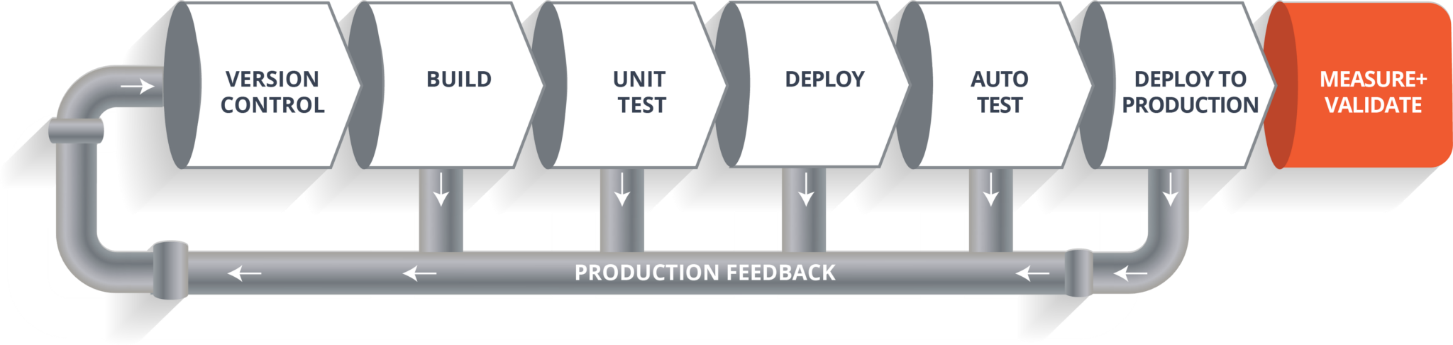
Once the code is deployed successfully, you can run another set of a sanity test. If everything is accepted, then it can be deployed to production.

### ****Deploy to Production:****

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

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

### ****Measure+Validate:****

So, this lifecycle continues until we get a code or a product which can be deployed in the production server where we measure and validate the code.

We have understood 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.

## [****Jenkins****](https://www.edureka.co/blog/what-is-jenkins/)****– The Ultimate CI Tool and Its Importance in CI CD Pipeline****

Our task is to automate the entire process, from the time the development team gives us the code and commits it to the time we get it into production.

Our task is to automate the pipeline in order to make the entire software development lifecycle on the Dev-ops mode/ automated mode. For this, they would need automation tools.



***Jenkins*** provides us with various interfaces and tools in order to automate the entire process.

So what happens, we have a git repository where the development team will commit the code. Then Jenkins takes over from there which is front-end tool where you can define your entire job or the task. Our job is to ensure the continuous integration and delivery process for that particular tool or for the particular application.

From Git, Jenkins pulls the code and then moves it to the **commit phase**, where the code is committed from every branch. Then Jenkins moves it into the **build phase** where we compile the code. If it is Java code, we use tools like maven in Jenkins and then compile that code, which we can be deployed to run a series of tests. These test cases are overseen by Jenkins again.

Then it moves on to the staging server to deploy it using **docker**. After a series of Unit Tests or sanity test, it moves to the production.

This is how the delivery phase is taken care by a tool called **Jenkins,** which automate everything. Now in order to deploy it, we will need an environment which will replicate the production environment, I.e., **Docker**.

**3🡪 Gitlab CI-CD =**

Git-lab CI/CD is a powerful tool built into gitlab that allows you to apply continuous Integration, Continuous Delivery, and Continuous Deployment to your software with no third party application or integration needed.

**The following points describe usage of GitLab CI/CD** −

* It is easy to learn, use and scalable.
* It is faster system which can be used for code deployment and development.
* You can execute the jobs faster by setting up your own runner (it is an application that processes the builds) with all dependencies which are pre-installed.
* GitLab CI solutions are economical and secure which are very flexible in costs as much as machine used to run it.
* It allows the project team members to integrate their work daily, so that the integration errors can be identified easily by an automated build.

**What you need to use GitLab CI-CD:-**

**1🡪** Application codebase hosted on Git Repository

**2🡪** Sequentially defined scripts in a YAML file

**3🡪** YAML file located in the root path of your Repository

**4🡪**Name of the file : gitlab-ci.yml

**How To write a YAML file :-**

**🡪** There is a simple pipeline configuration.Youare using Docker image based on Alpine Linux.You have two builds, **build A builds** “ something” **and B builds “**something else”.

**🡪**  You need to configure this pipeline using a YAML file called

.gitlab-ci.yml.

**Why use GitLab CI/CD ?**

**1🡪** Higher Visibility

**2🡪** Wholesome CI/cd Solution

**3🡪** Easy Rollback of Changes

**4🡪**Automated Triggers

Step 1: Install GitLab Runner

[https://docs.gitlab.com/runner/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmFWaU5odXRpVXEtVmF2a090LU9YZWFFM1V2UXxBQ3Jtc0treDBpUVFfOEYtQURFNmZoelh6Y1JCWkVSQUhJaVhoXzlrQzh1SENoQkRLdnd0MThQQVJmMGlFeVRNTHkzcUFiZHlMakFqUlBIZjVMM1ZZRERVU01teEktMVVlcDJCY2FtMFJQYkpxUHEtcUFaOHhqWQ&q=https%3A%2F%2Fdocs.gitlab.com%2Frunner%2F)​

[https://docs.gitlab.com/runner/instal...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbTRoeTB3VHRnX3kxWDBDQ0ZTYXRjd2xUVW9Yd3xBQ3Jtc0tuM0R0YVV2bE9mX3l3SlJsYmtmaWtqbVVTR0RkV2haVnY0ZktBQUJYb25tMk9FdnlNSzNCMDNvb3pvSmlRYVFHMDBKQlc3UHZjR2pfQXRWcFVnZTRaaWJsVkstM1BpZl9iNHVHUFFCYU4zU0VoR1ZOSQ&q=https%3A%2F%2Fdocs.gitlab.com%2Frunner%2Finstall%2Findex.html)​

gitlab-runner.exe install

gitlab-runner --version

Step 2: Register GitLab Runner

(process to bind runner with gitlab instance)

[https://docs.gitlab.com/runner/regist...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEFJVzlydHJrZGxsaW5HaEV2ZkNuWm5pTGlTQXxBQ3Jtc0tsVU53U0dielZWUTZ0SlVERDljVXN1TjVoNkt6cTVUOGlINlYwbW1WaVhSeXRubXFmd3l0WUQ3OHhOWmR4bmJsa3FTZ2VRb2V3OVhOc0ZNODc0ZnJuWk5pdEVQTlM4ZEU3a29acDhjTXJoc0F2dnlPQQ&q=https%3A%2F%2Fdocs.gitlab.com%2Frunner%2Fregister%2Findex.html)​

gitlab-runner.exe register

Step 3: Start GitLab Runner

gitlab-runner.exe start

Step 4: Check runner is started in the project

Git-lab CI (Continuous Integration) service is a part of GitLab that build and test the software whenever developer pushes code to application. GitLab CD (Continuous Deployment) is a software service that places the changes of every code in the production which results in every day deployment of production.

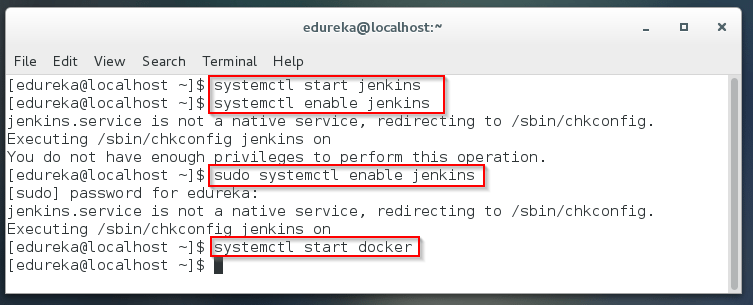
The following points describe usage of GitLab CI/CD −

* It is easy to learn, use and scalable.
* It is faster system which can be used for code deployment and development.
* You can execute the jobs faster by setting up your own runner (it is an application that processes the builds) with all dependencies which are pre-installed.
* GitLab CI solutions are economical and secure which are very flexible in costs as much as machine used to run it.
* It allows the project team members to integrate their work daily, so that the integration errors can be identified easily by an automated build.

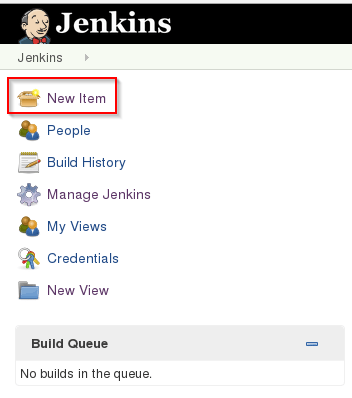
**Hands-On: Building CI CD Pipeline Using Docker and Jenkins**

**Step 1:** Open your terminal in your VM. Start Jenkins and Docker using the commands “**systemctl start jenkins**“, “**systemctl enable jenkins**“, “**systemctl start docker**“.

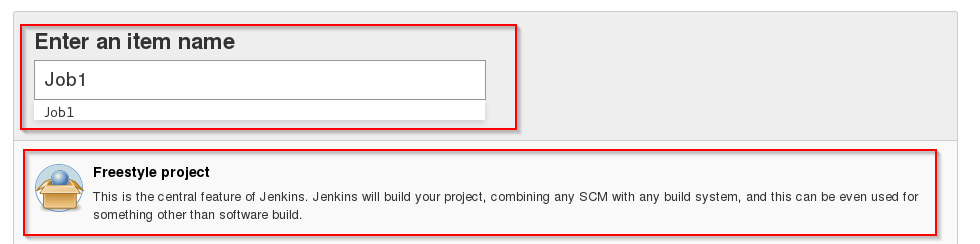
***Note:****Use****sudo****before the commands if it display “privileges error”.*



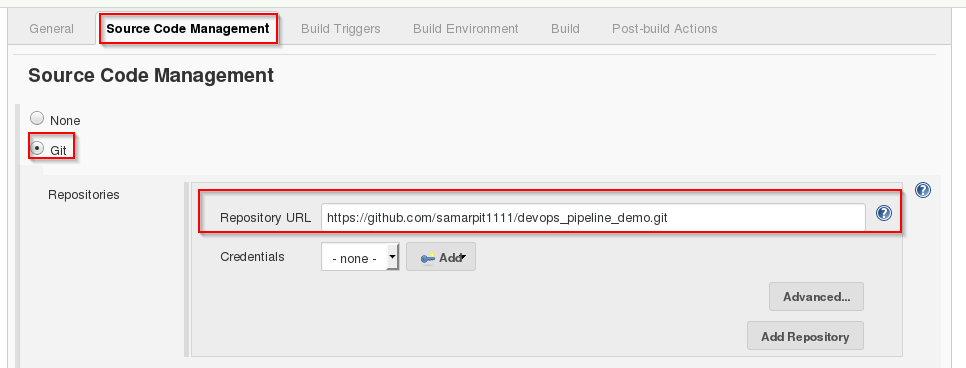
**Step 2:** Open your Jenkins on your specified port. Click on **New Item**to create a Job.



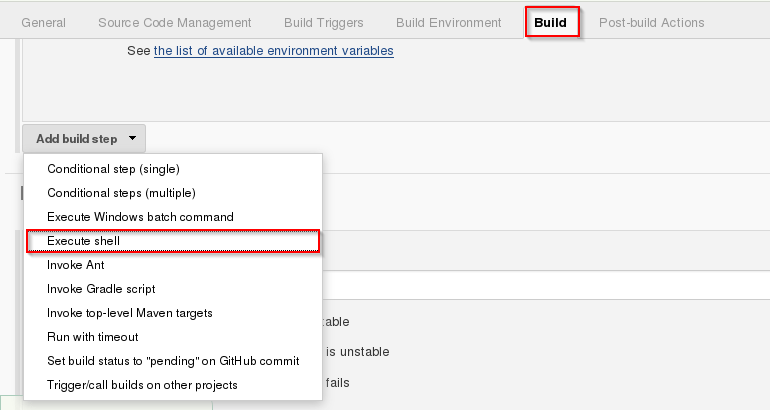
**Step 3:** Select **freestyle** project and provide the item name (here I have given Job1) and click OK.



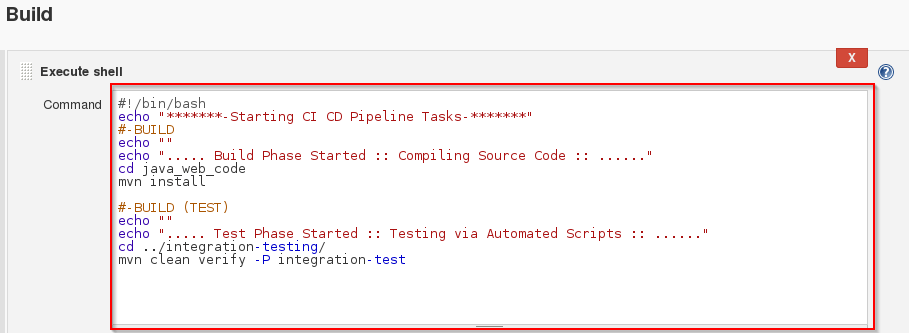
**Step 4:** Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.



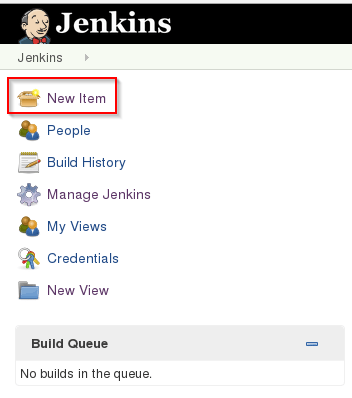
**Step 5:** Then click on **Build->Select Execute Shell**.



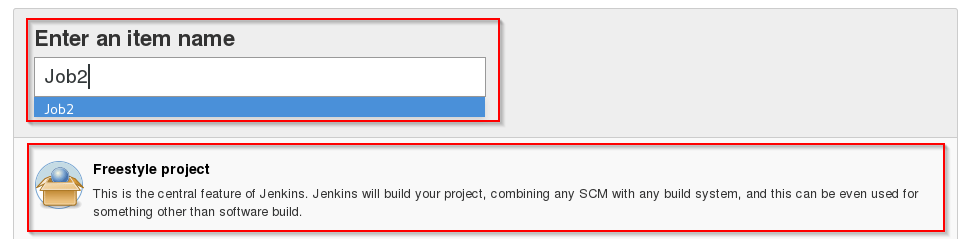
**Step 6:** Provide the shell commands. Here it will build the archive file to get a war file. After that, it will get the code which is already pulled and then it uses maven to install the package. So, it simply installs the dependencies and compiles the application.



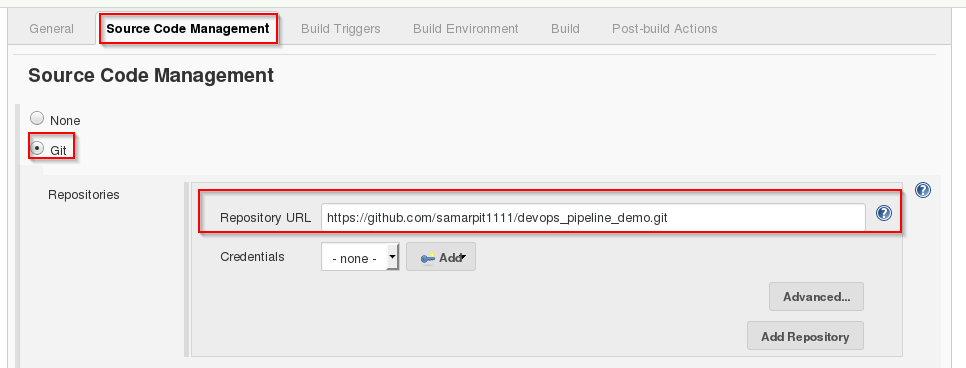
**Step 7:** Create the new **Job** by clicking on New Item.



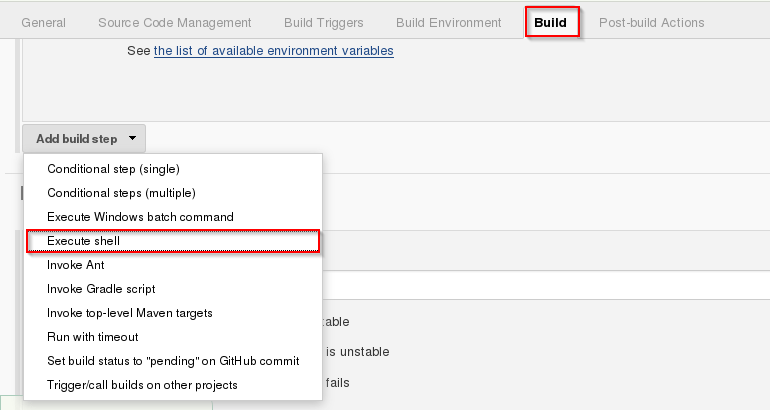
**Step 8:** Select **freestyle** project and provide the item name (here I have given Job2) and click on OK.



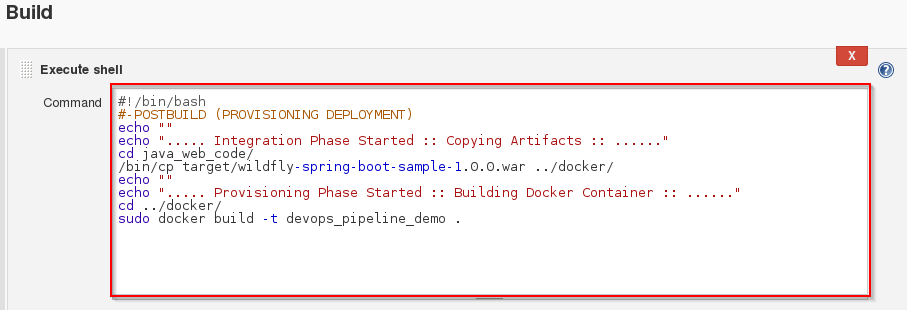
**Step 9:** Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.



**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.

**DevOps Training**

**[DEVOPS CERTIFICATION TRAINING](https://www.edureka.co/devops-certification-training" \o "DevOps Certification Training" \t "_blank)**

**[DevOps Certification Training](https://www.edureka.co/devops-certification-training" \o "DevOps Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/devops-certification-training" \o "DevOps Certification Training" \t "_blank)*

**[5](https://www.edureka.co/devops-certification-training" \o "DevOps Certification Training" \t "_blank)**[(93673)](https://www.edureka.co/devops-certification-training" \o "DevOps Certification Training" \t "_blank)

**[AWS CERTIFIED DEVOPS ENGINEER TRAINING](https://www.edureka.co/aws-certified-devops-training" \o "AWS Certified DevOps Engineer Training " \t "_blank)**

**[AWS Certified DevOps Engineer Training](https://www.edureka.co/aws-certified-devops-training" \o "AWS Certified DevOps Engineer Training " \t "_blank)**

*[Reviews](https://www.edureka.co/aws-certified-devops-training" \o "AWS Certified DevOps Engineer Training " \t "_blank)*

**[5](https://www.edureka.co/aws-certified-devops-training" \o "AWS Certified DevOps Engineer Training " \t "_blank)**[(4039)](https://www.edureka.co/aws-certified-devops-training" \o "AWS Certified DevOps Engineer Training " \t "_blank)

**[KUBERNETES CERTIFICATION TRAINING](https://www.edureka.co/kubernetes-certification" \o "Kubernetes Certification Training" \t "_blank)**

**[Kubernetes Certification Training](https://www.edureka.co/kubernetes-certification" \o "Kubernetes Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/kubernetes-certification" \o "Kubernetes Certification Training" \t "_blank)*

**[5](https://www.edureka.co/kubernetes-certification" \o "Kubernetes Certification Training" \t "_blank)**[(6875)](https://www.edureka.co/kubernetes-certification" \o "Kubernetes Certification Training" \t "_blank)

**[GIT AND GITHUB](https://www.edureka.co/git-github-sp" \o "Git and GitHub" \t "_blank)**

**[Git and GitHub](https://www.edureka.co/git-github-sp" \o "Git and GitHub" \t "_blank)**

*[Reviews](https://www.edureka.co/git-github-sp" \o "Git and GitHub" \t "_blank)*

**[5](https://www.edureka.co/git-github-sp" \o "Git and GitHub" \t "_blank)**[(3590)](https://www.edureka.co/git-github-sp" \o "Git and GitHub" \t "_blank)

**[DOCKER CERTIFICATION TRAINING](https://www.edureka.co/docker-training" \o "Docker Certification Training" \t "_blank)**

**[Docker Certification Training](https://www.edureka.co/docker-training" \o "Docker Certification Training" \t "_blank)**

*[Reviews](https://www.edureka.co/docker-training" \o "Docker Certification Training" \t "_blank)*

**[5](https://www.edureka.co/docker-training" \o "Docker Certification Training" \t "_blank)**[(5651)](https://www.edureka.co/docker-training" \o "Docker Certification Training" \t "_blank)

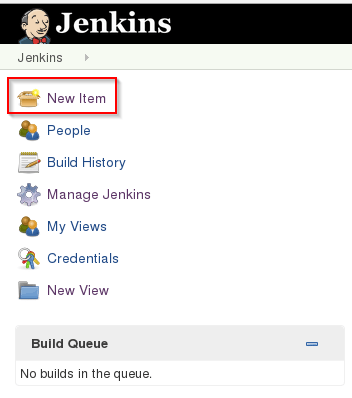
**[CONTINUOUS INTEGRATION WITH JENKINS CERTIFICATION TRAINING](https://www.edureka.co/jenkins" \o "Continuous Integration with Jenkins Certification Training" \t "_blank)**

**[Continuous Integration with Jenkins Certification Training](https://www.edureka.co/jenkins" \o "Continuous Integration with Jenkins Certification Training" \t "_blank)**

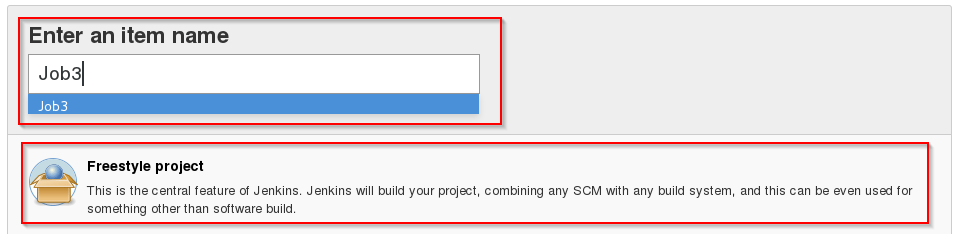
*[Reviews](https://www.edureka.co/jenkins" \o "Continuous Integration with Jenkins Certification Training" \t "_blank)*

**[5](https://www.edureka.co/jenkins" \o "Continuous Integration with Jenkins Certification Training" \t "_blank)**[(8046)](https://www.edureka.co/jenkins" \o "Continuous Integration with Jenkins Certification Training" \t "_blank)

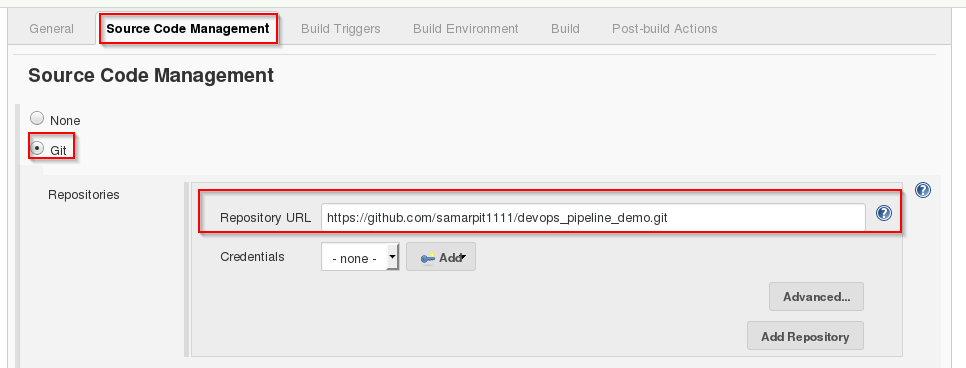
Next



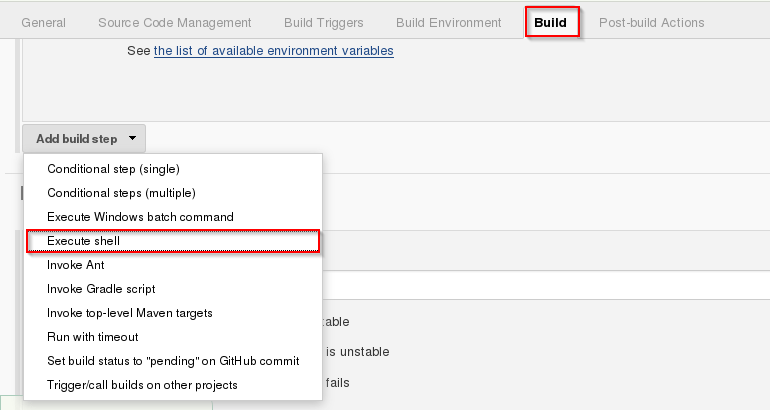
**Step 13:** Select **freestyle** project and provide the item name (here I have given Job3) and click on OK.



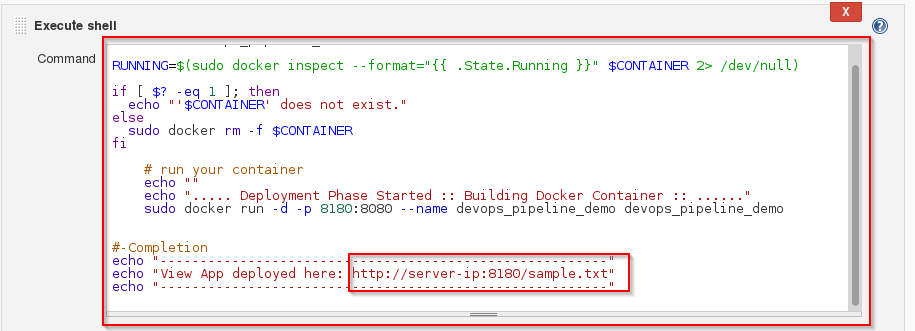
**Step 14:** Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.



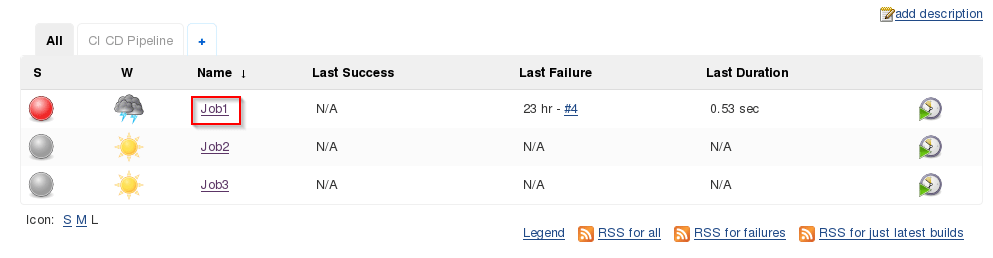
**Step 15:** Then click on **Build->Select Execute Shell**.



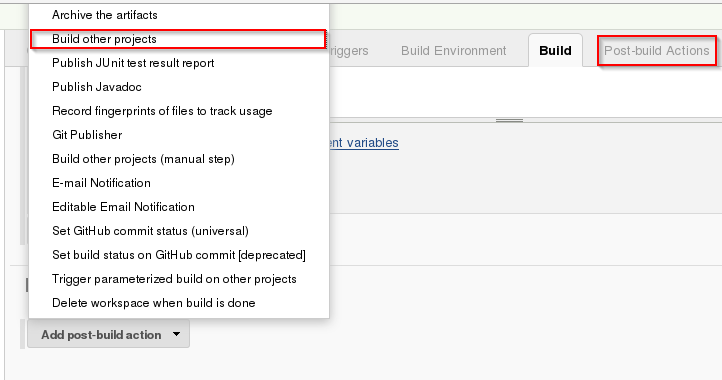
**Step 16:** Provide the shell commands. Here it will check for the Docker Container file and then deploy it on port number 8180. Click on Save button.



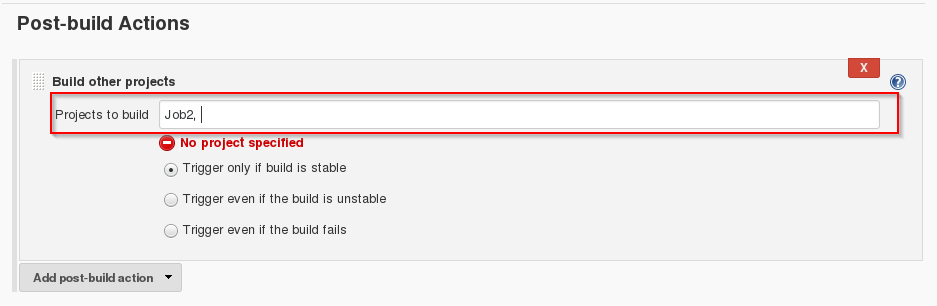
**Step 17:** Now click on **Job1 -> Configure**.



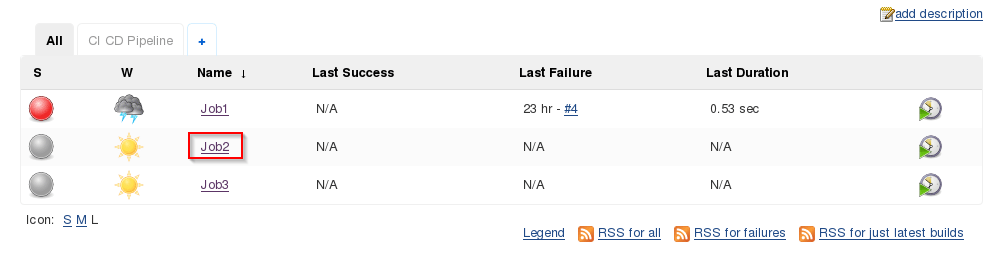
**Step 18:** Click on **Post-build Actions -> Build other projects**.

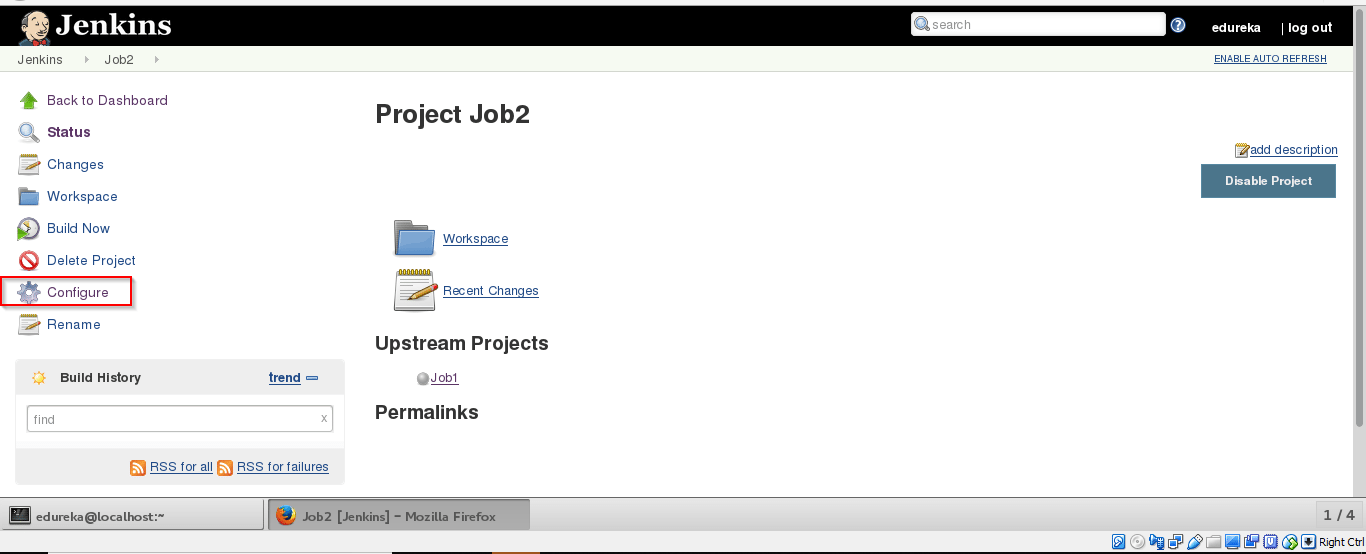


**Step 19:** Provide the project name to build after Job1 (here is Job2) and then click on **Save**.

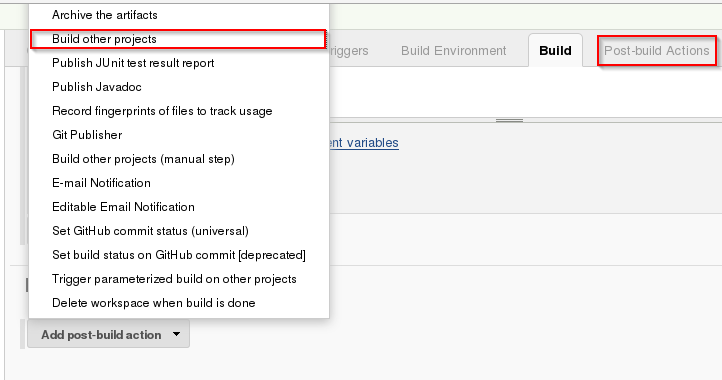


**Step 20:**Now click on **Job2 -> Configure**.

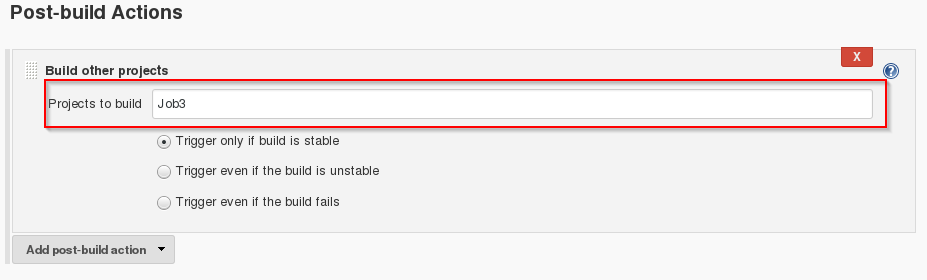




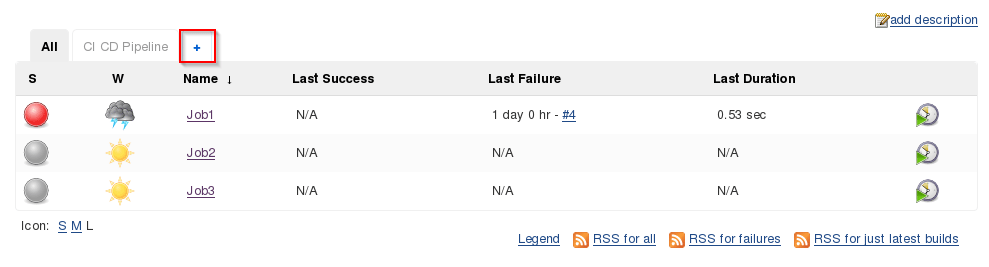
**Step 21:** Click on **Post-build Actions -> Build other projects**.



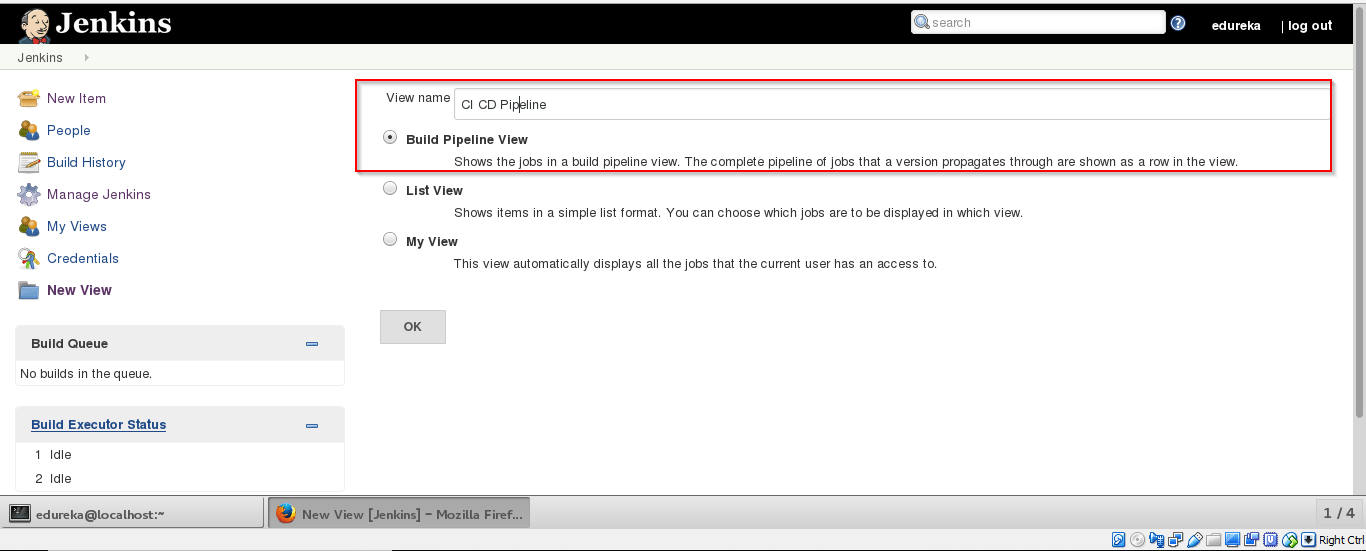
**Step 22:** Provide the project name to build after Job2 (here is Job3) and then click on **Save**.



**Step 23:** Now we will be creating a Pipeline view. Click on ‘+’ sign.



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

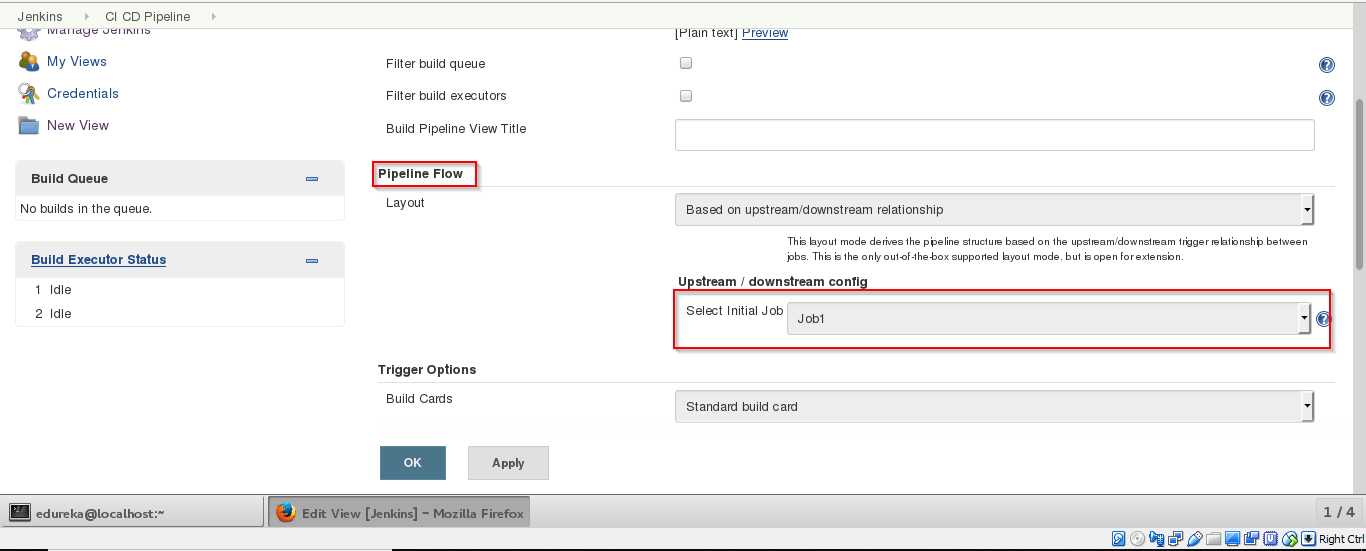


[[](https://www.edureka.co/devops-certification-training)](https://www.edureka.co/devops-certification-training" \t "_blank)

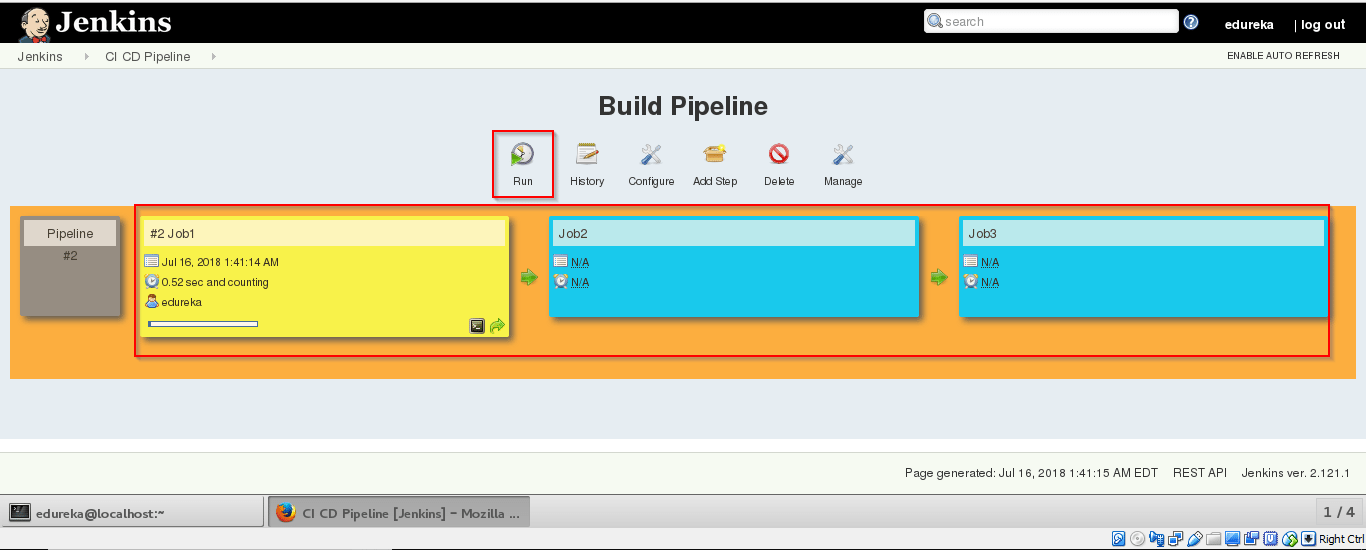
**[DevOps Certification Training](https://www.edureka.co/devops-certification-training" \t "_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/devops-certification-training" \t "_blank)

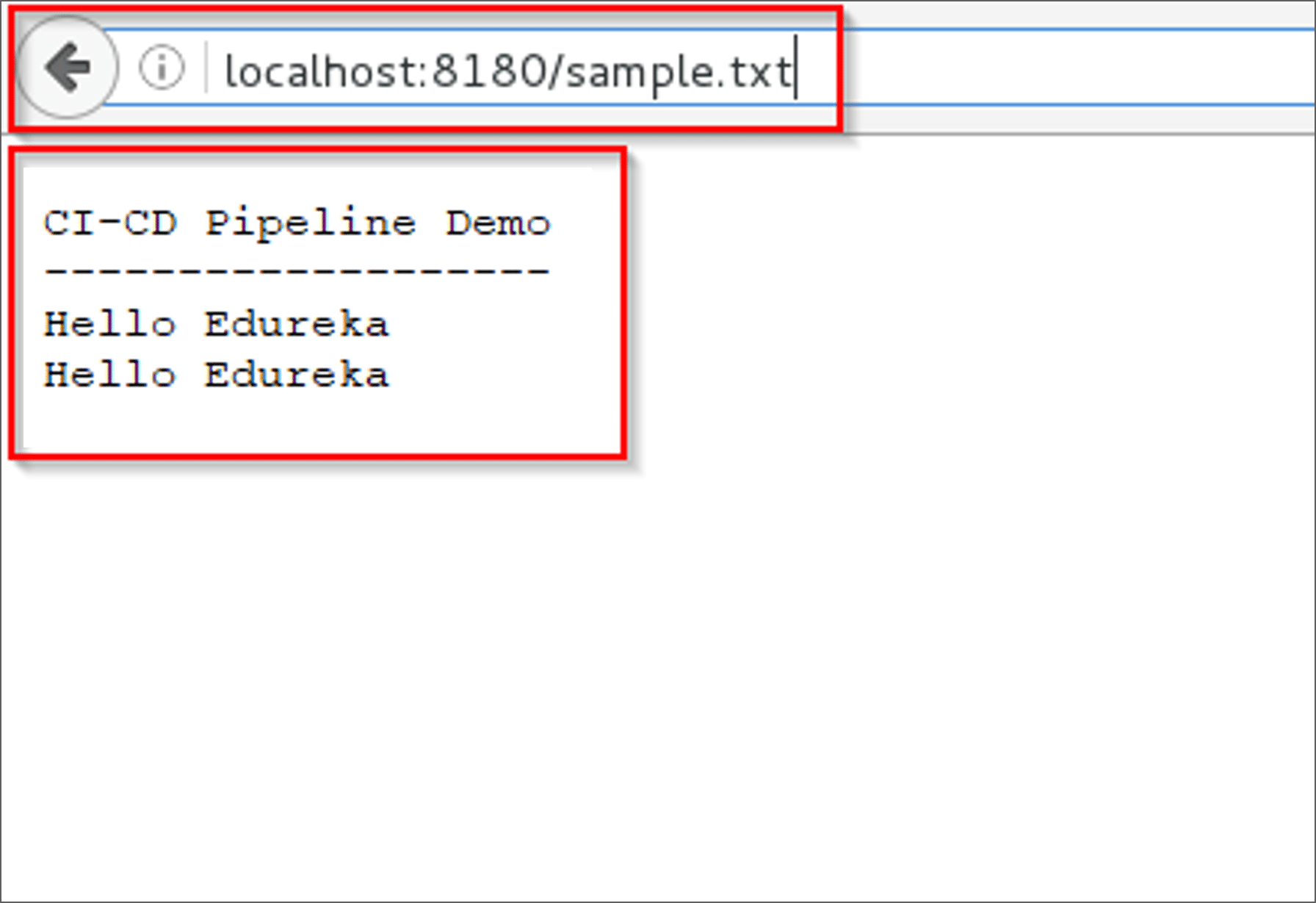
**Step 25:** Select the **initial** **Job** (here I have provided Job1) and click on OK.



**Step 26:** Click on **Run**button to start the 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 CI CD Pipeline using Docker and Jenkins.