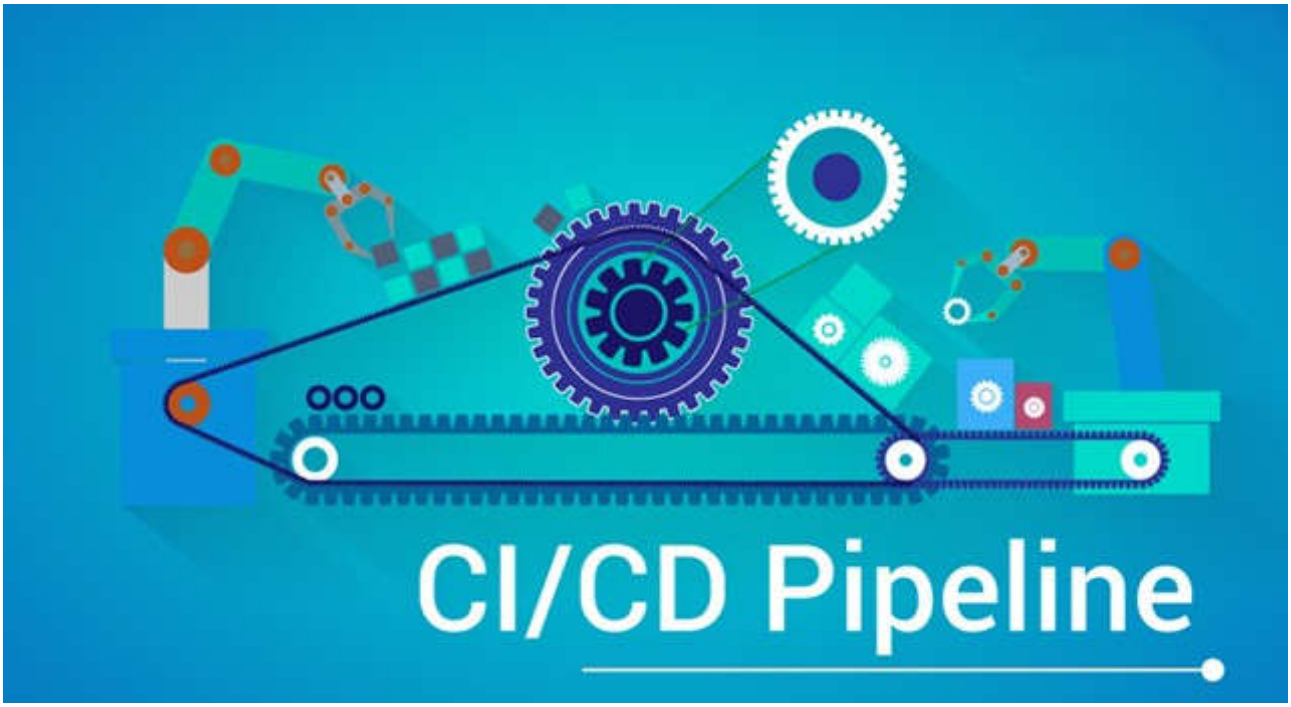
CI CD Pipeline: Learn How to Setup a CI CD Pipeline From Scratch

Saurabh Kulshrestha
Aug 9, 2018 · 10 min read



CI CD Pipeline – Edureka

CI CD Pipeline implementation or the Continuous Integration/Continuous Deployment software is the backbone of the modern DevOps environment. You can find the requirement of *Continuous Integration & Continuous Deployment skills* in various job roles such as Data Engineer, Cloud Architect, Data Scientist, etc. CI/CD bridges the gap between development and operations teams by automating build, test, and deployment of applications. In this blog, we will know What is CI CD pipeline and how it works.

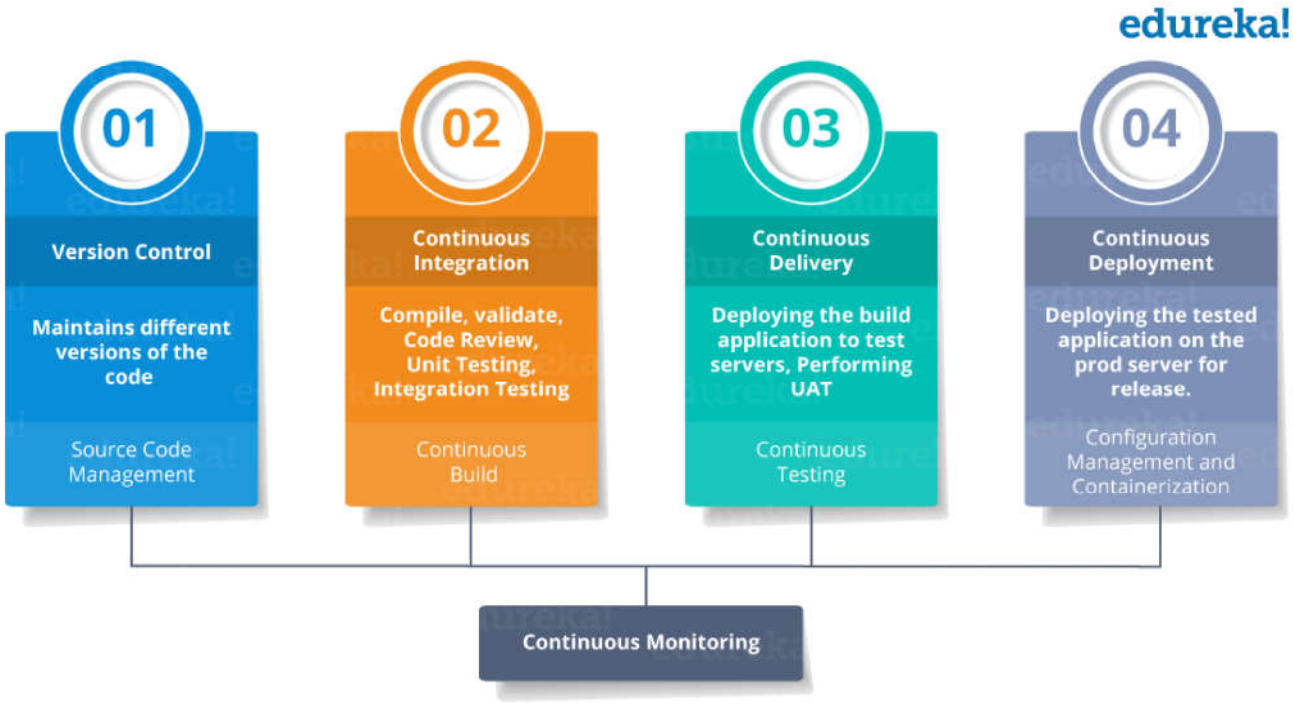
Before moving onto the CI CD pipeline's working, let's start by understanding DevOps.

What is DevOps?

DevOps is a software development approach which involves continuous development, continuous testing, continuous integration, continuous deployment and continuous monitoring of the software throughout its development life cycle. This is exactly the process adopted by all the top companies to develop high-quality software and shorter development life cycles, resulting in greater customer satisfaction, something that every company wants.

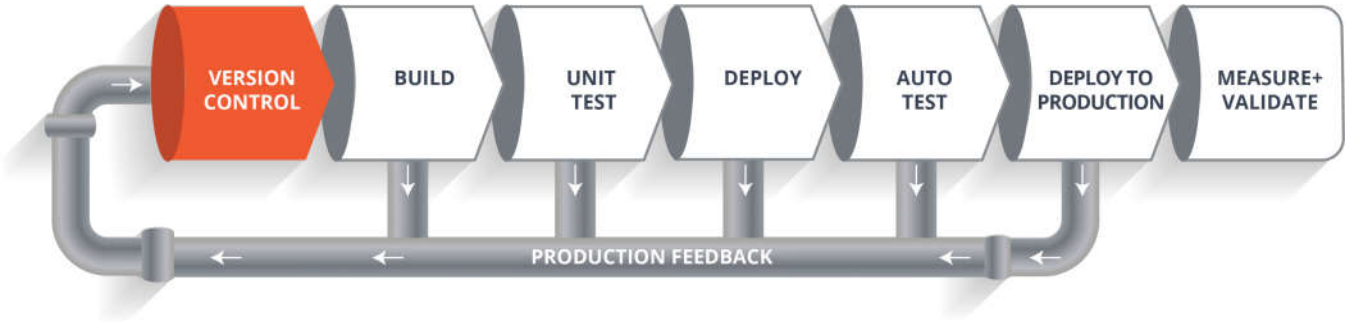
DevOps Stages

Your understanding of what is DevOps is incomplete without learning about its life cycle. Let us now look at the DevOps lifecycle and explore how they are related to the software development stages.



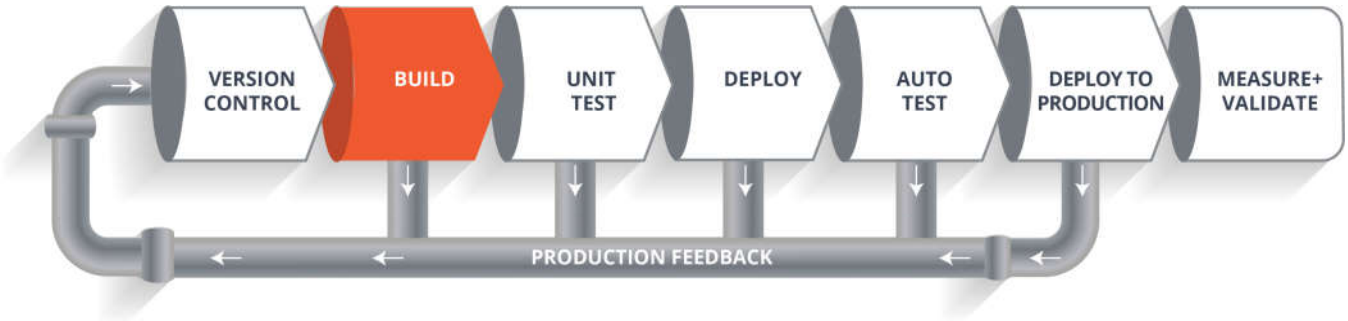
What is CI CD Pipeline?

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 work.



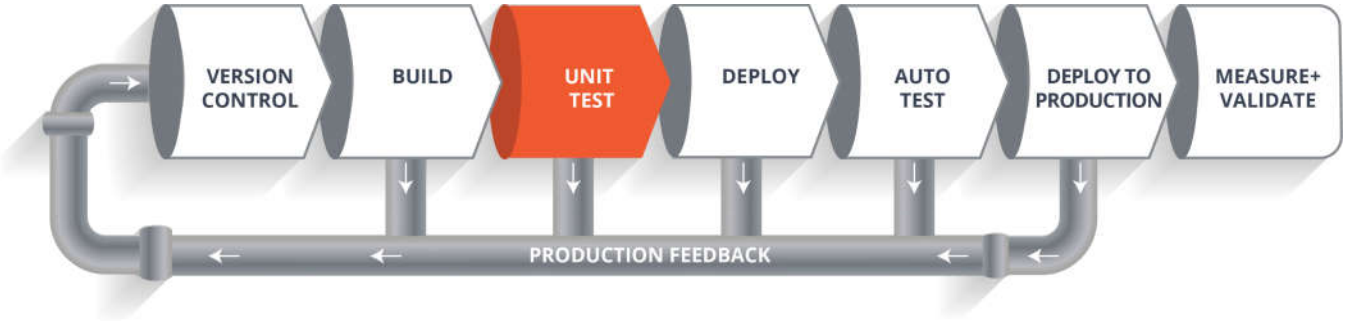
The above pipeline is a logical demonstration of how software will move along the various phases or stages in this lifecycle, before it is delivered to the customer or before it is live on production.

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.



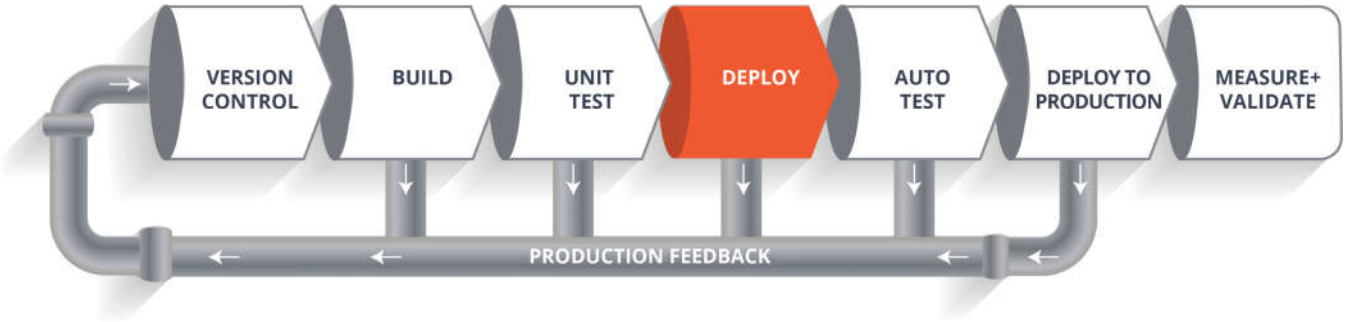
Suppose we have a Java code and it needs to be compiled before execution. So, through the version control phase, it again goes to build phase where it gets compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it.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

Testing Phase:



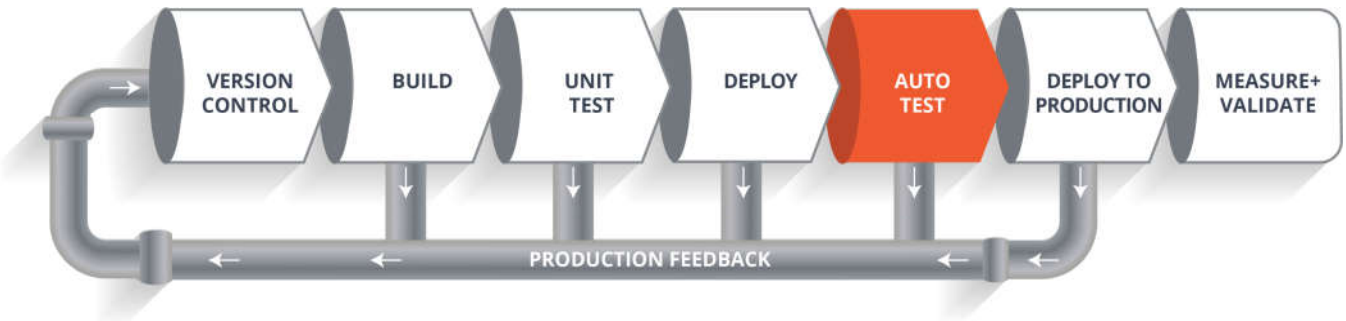
Once the build phase is over, then you move on to the **testing phase.** In this phase, we have various kinds of testing, one of them is the *unit test* (where you test the chunk/unit of software or for its sanity test).

Deploy Phase:



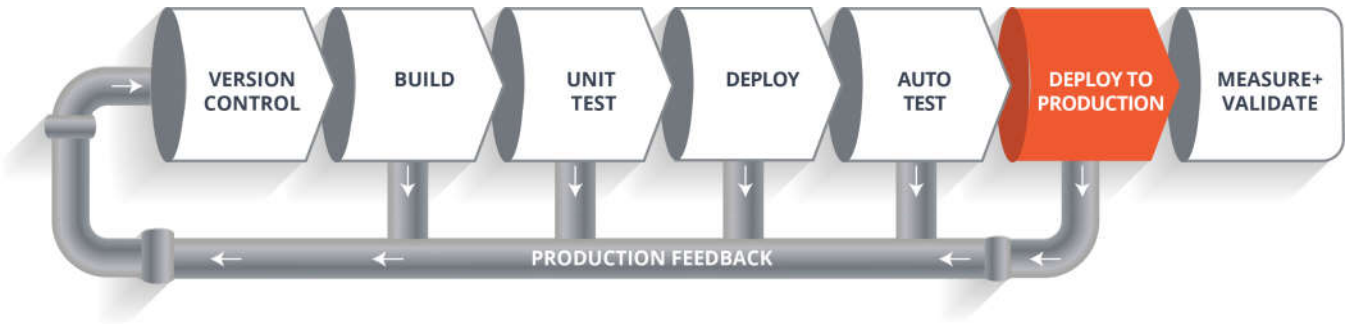
When the test is completed, you move on to the **deploy phase**, where you deploy it into a staging or a test server. Here, you can view the code or you can view the app in a simulator.

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.

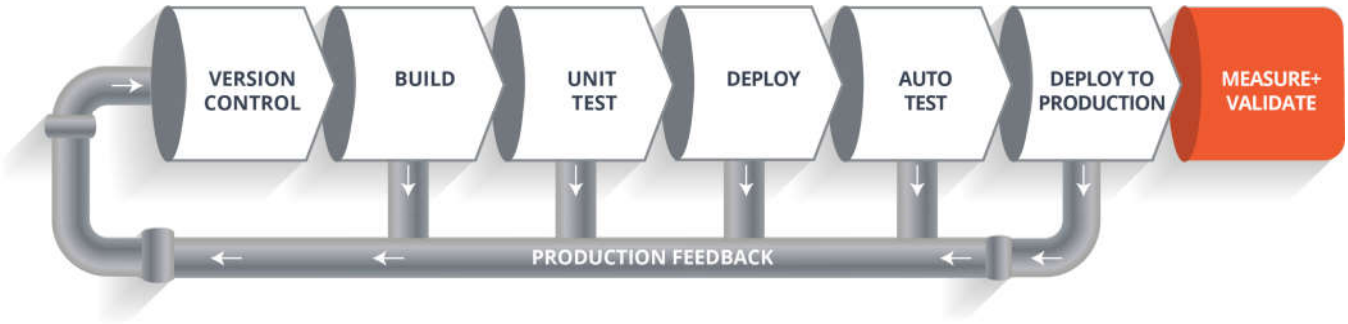
Deploy to Production:



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:



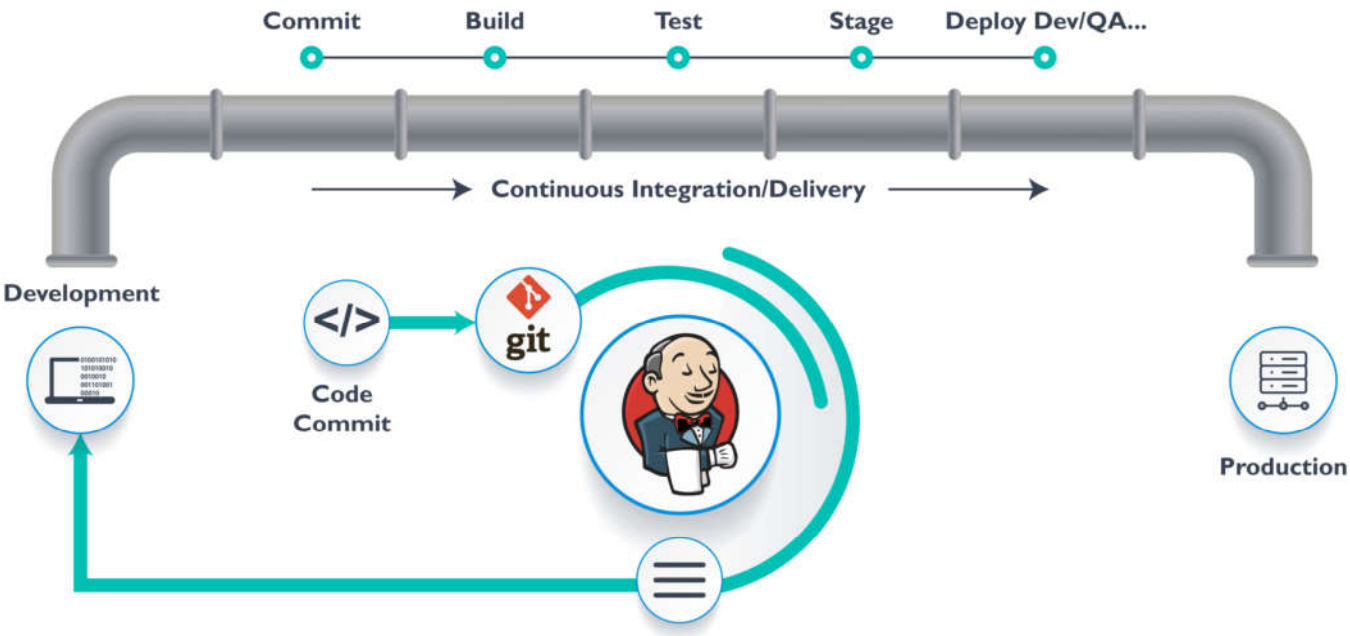
So, this lifecycle continues until we get a code or a product which can be deployed in the production server where we measure and validate the code.

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 — 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.

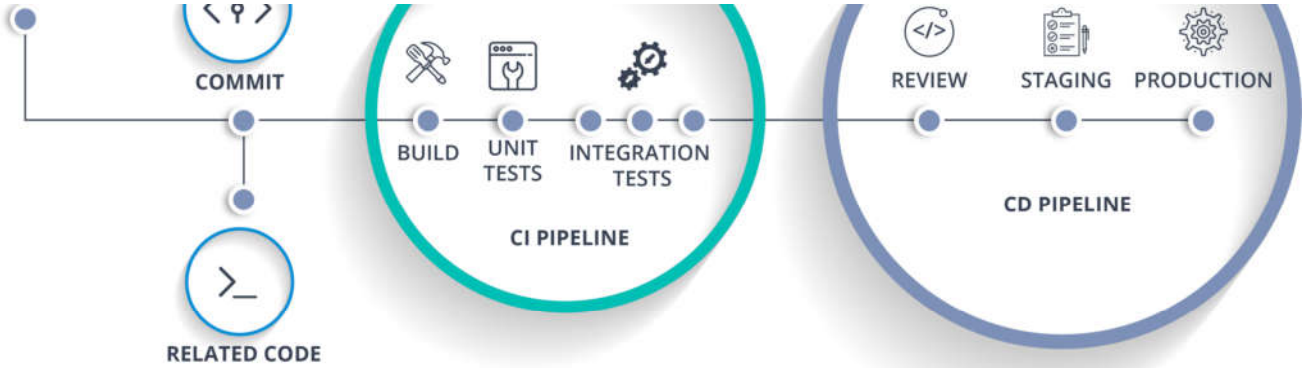
So what happens, we have a git repository where the development team will commit the code. Then Jenkins takes over from there which is a front-end tool where you can define your entire job or the task. Our job is to ensure the continuous integration and delivery process for that particular tool or for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

Docker



Docker is just like a virtual environment in which we can create a server. It takes a few seconds to create an entire server and deploy the artifacts which we want to test. But here the question arises,

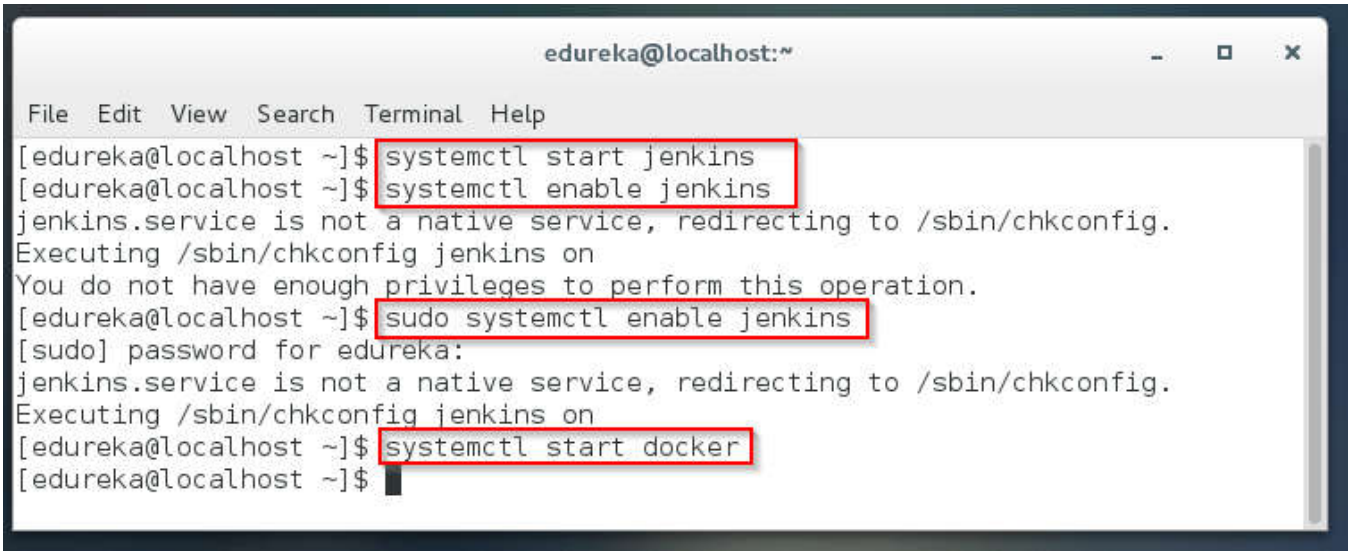
Why do we use docker?

As said earlier, you can run the entire cluster in a few seconds. We have storage registry for images where you build your image and store it forever. You can use it anytime in any environment which can replicate itself.

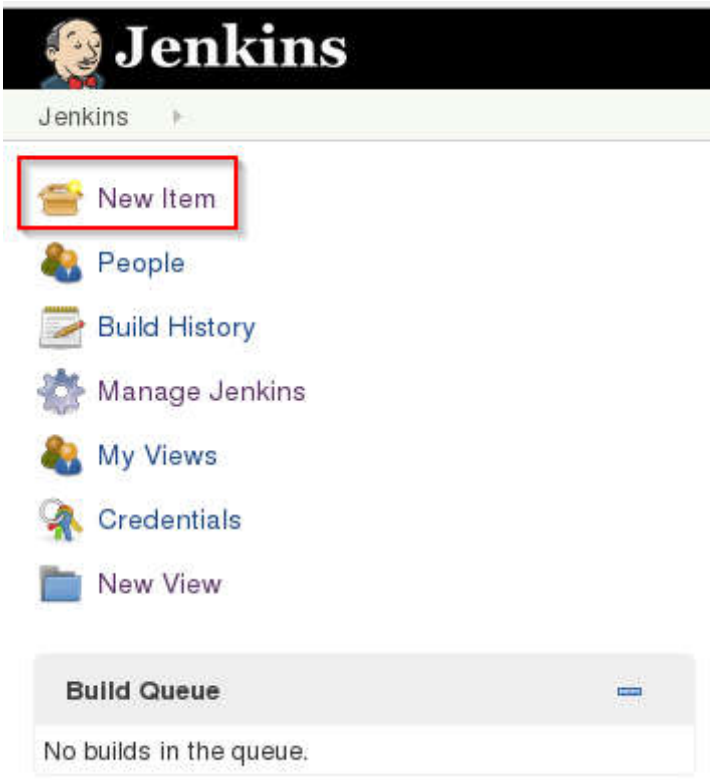
Hands–On: Building CI CD Pipeline Using Docker and Jenkins

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

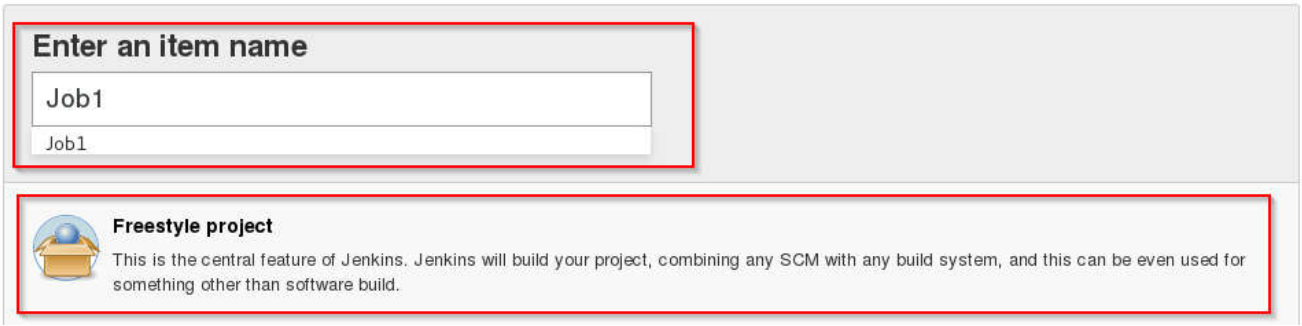
Note: *Use* ***sudo*** *before the commands if it display "privileges error".*



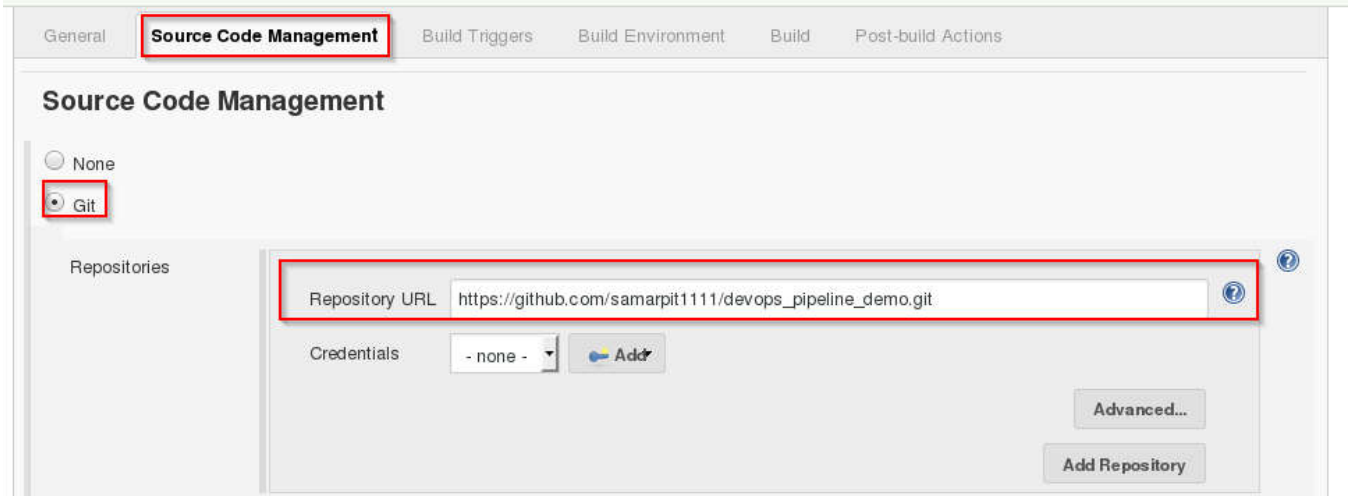
Step 2: Open your Jenkins on your specified port. Click on **New Item** to create a Job.



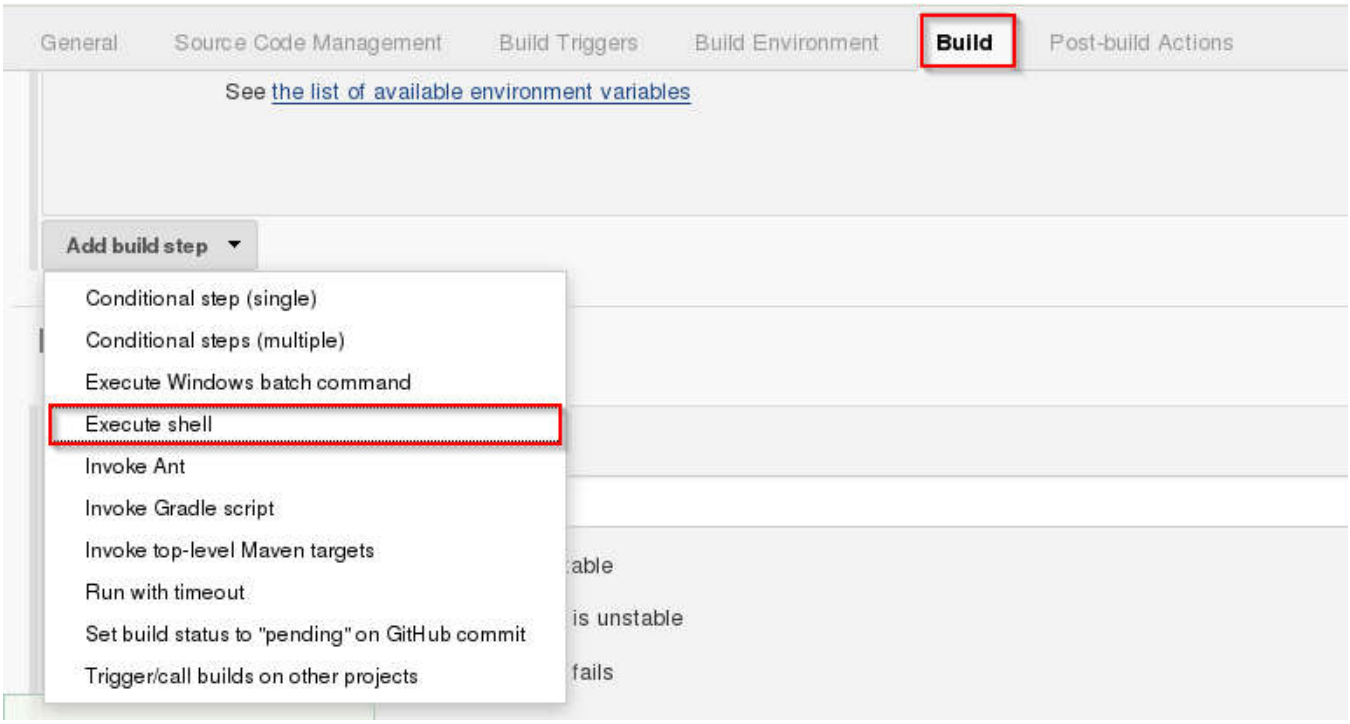
Step 3: Select **freestyle** project and provide the item name (here I have given Job1) and click OK.



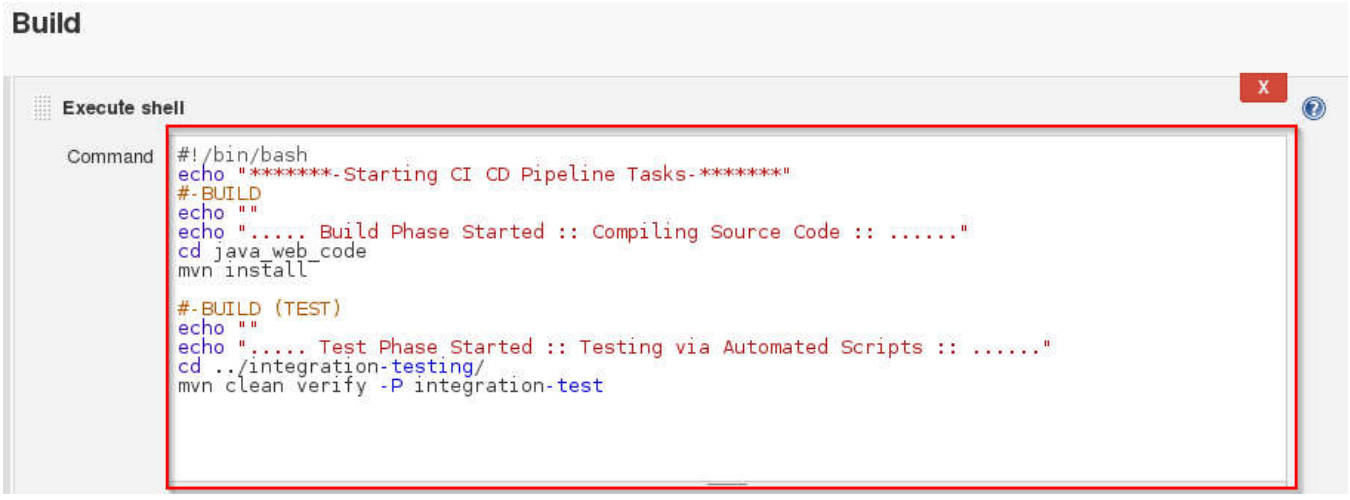
Step 4: Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.



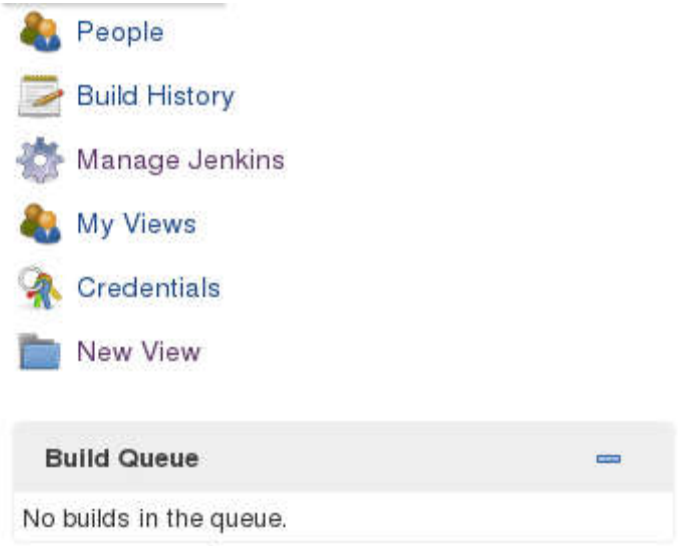
Step 5: Then click on **Build->Select Execute Shell.**



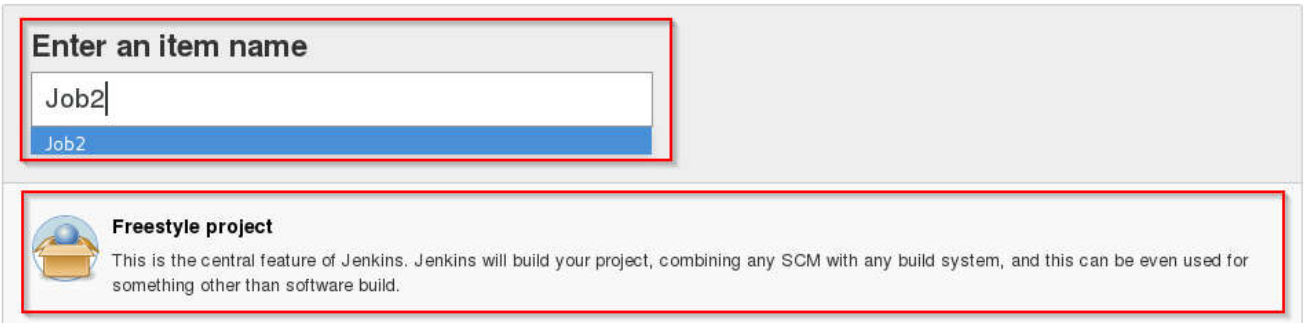
Step 6: Provide the shell commands. Here it will build the archive file to get a war file. After that, it will get the code which is already pulled and then it uses maven to install the package. So, it simply installs the dependencies and compiles the application.



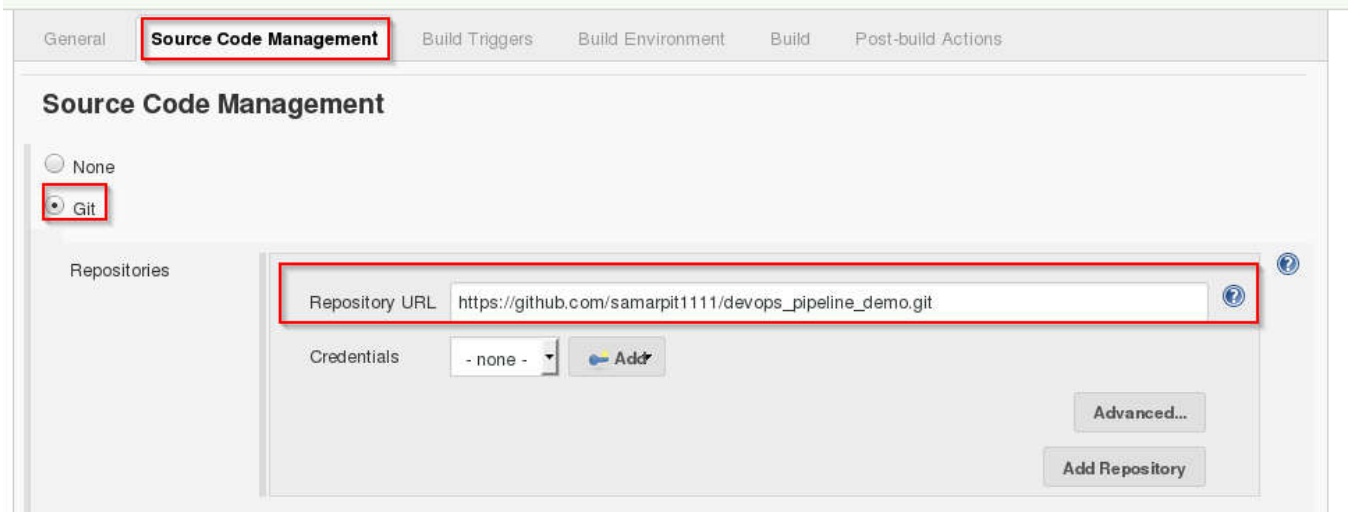
Step 7: Create the new **Job** by clicking on New Item.



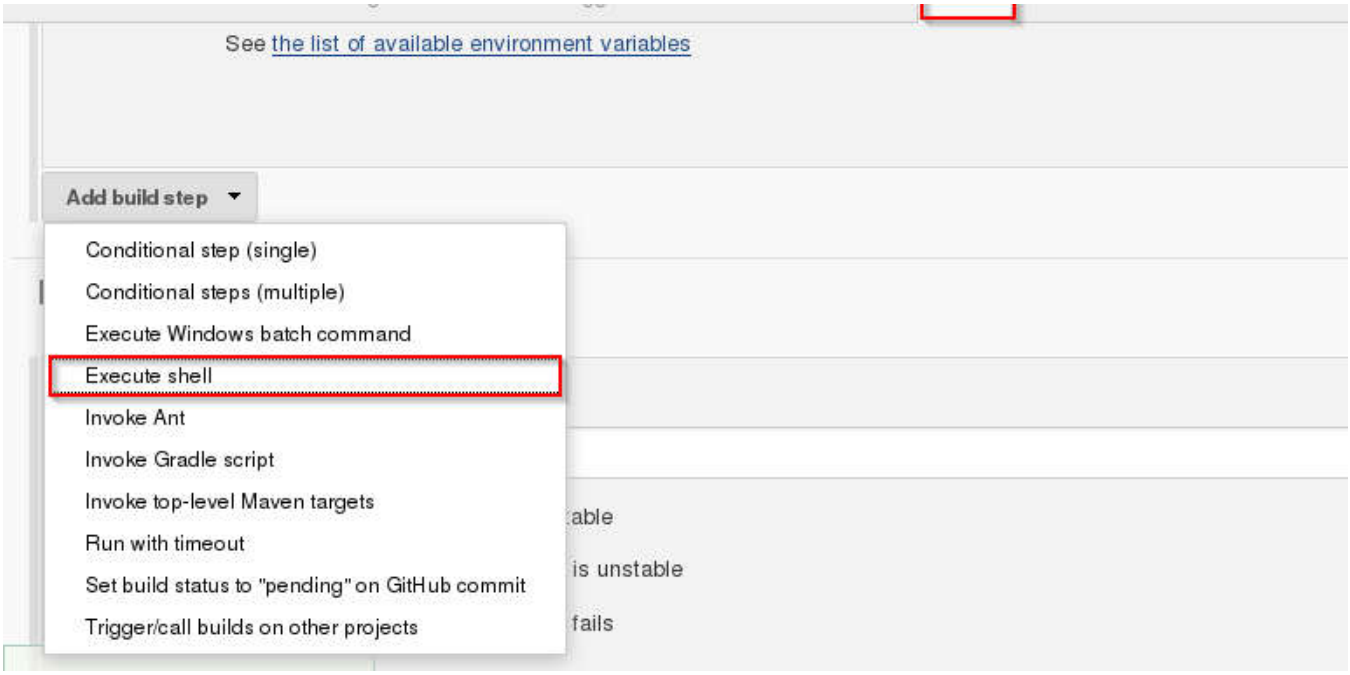
Step 8: Select **freestyle** project and provide the item name (here I have given Job2) and click on OK.



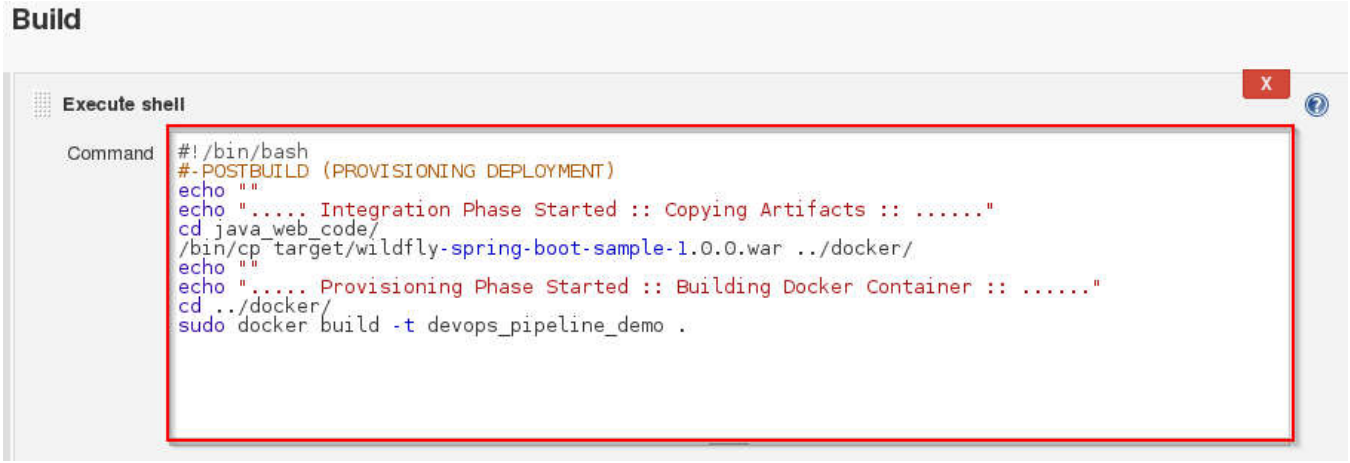
Step 9: Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.



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

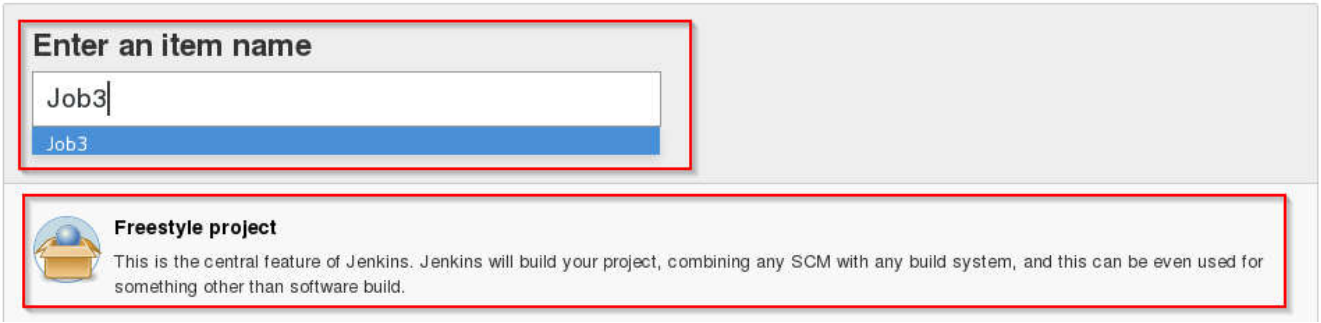


Step 11: Provide the shell commands. Here it will start the integration phase and **build**the Docker Container.

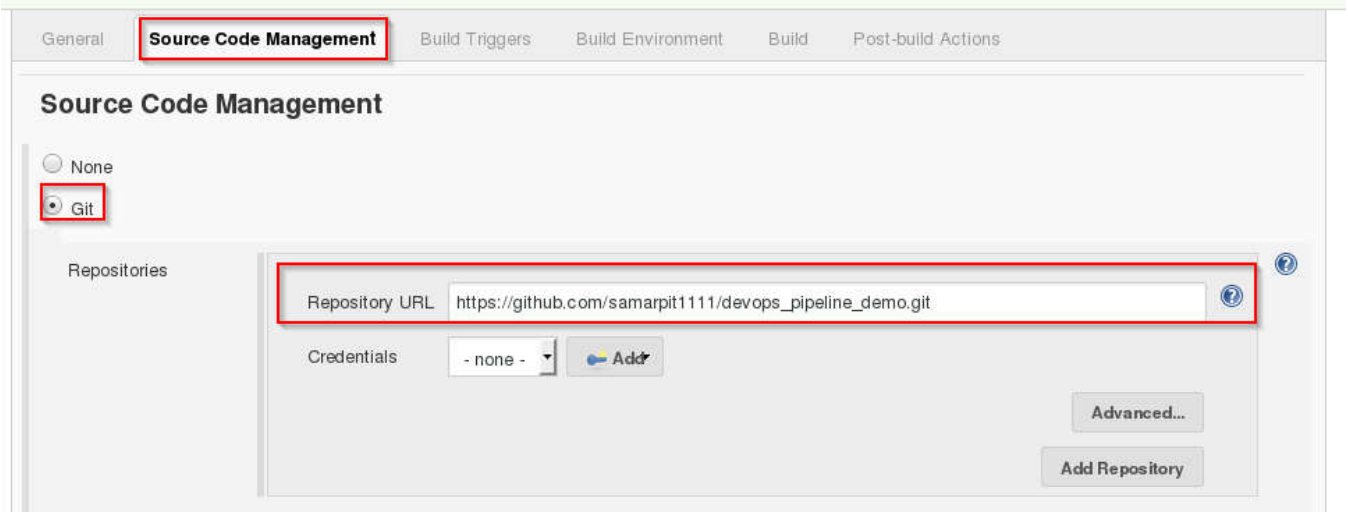


Step 12: Create the new **Job** by clicking on New Item.

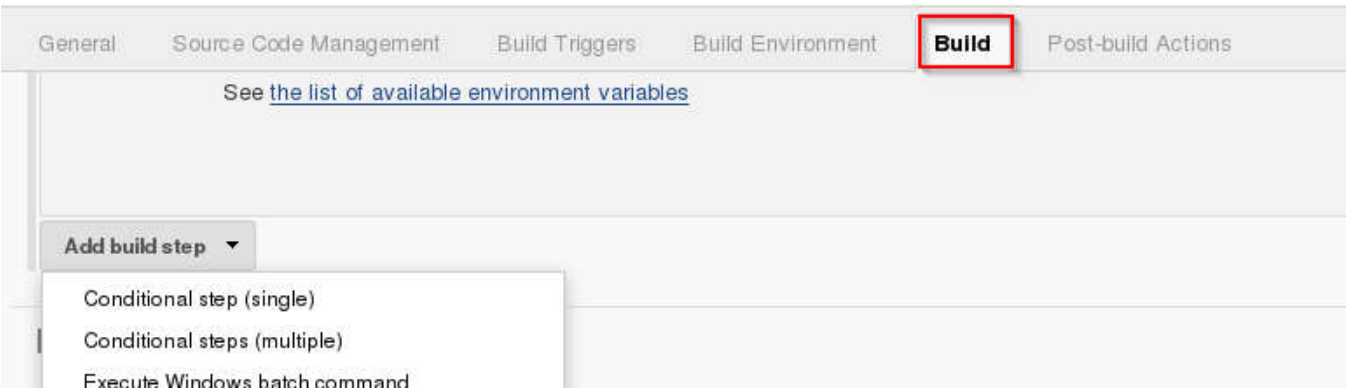
Step 13: Select **freestyle** project and provide the item name (here I have given Job3) and click on OK.



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

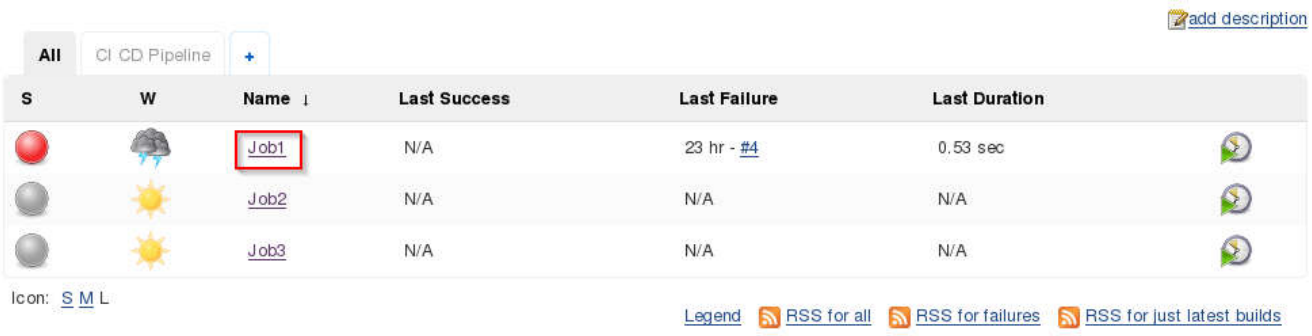


Step 15: Then click on **Build->Select Execute Shell.**

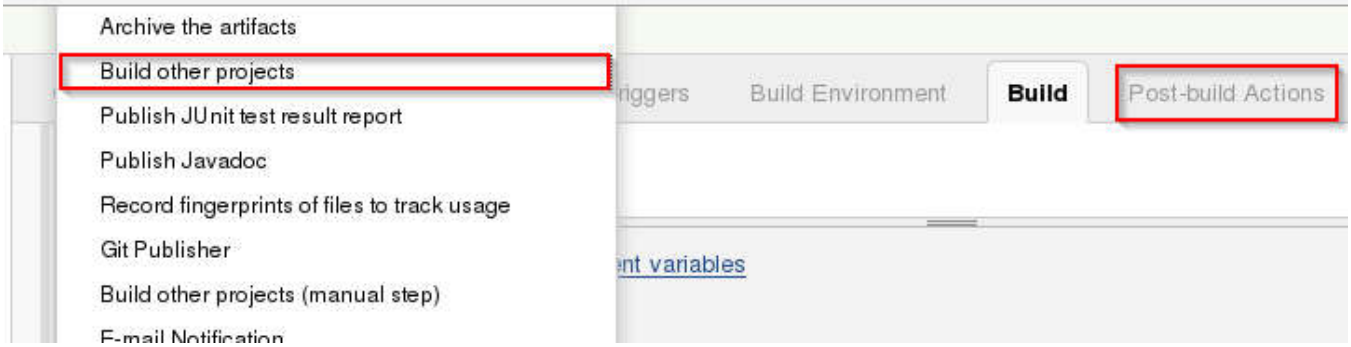


Step 16: Provide the shell commands. Here it will check for the Docker Container file and then deploy it on port number 8180. Click on Save button.

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

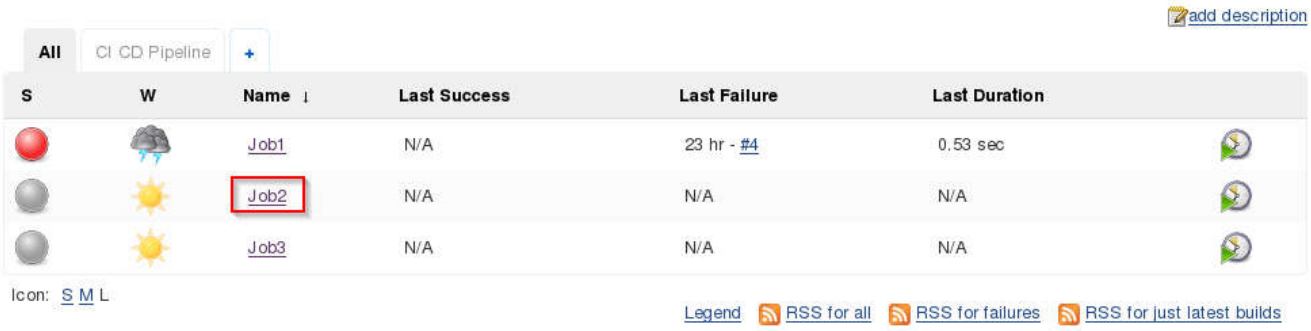


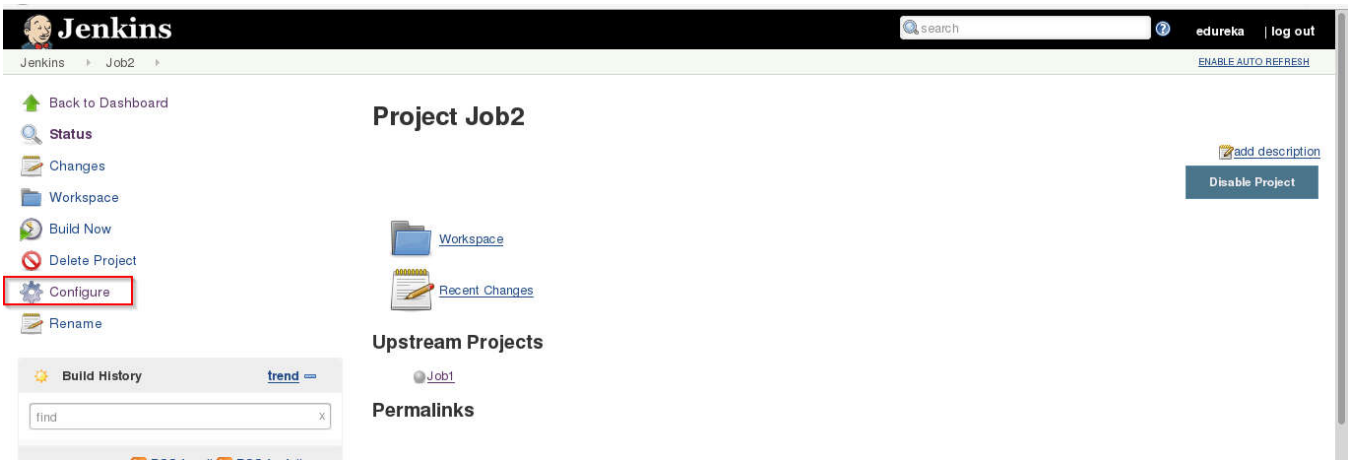
Step 18: Click on **Post-build Actions -> Build other projects.**



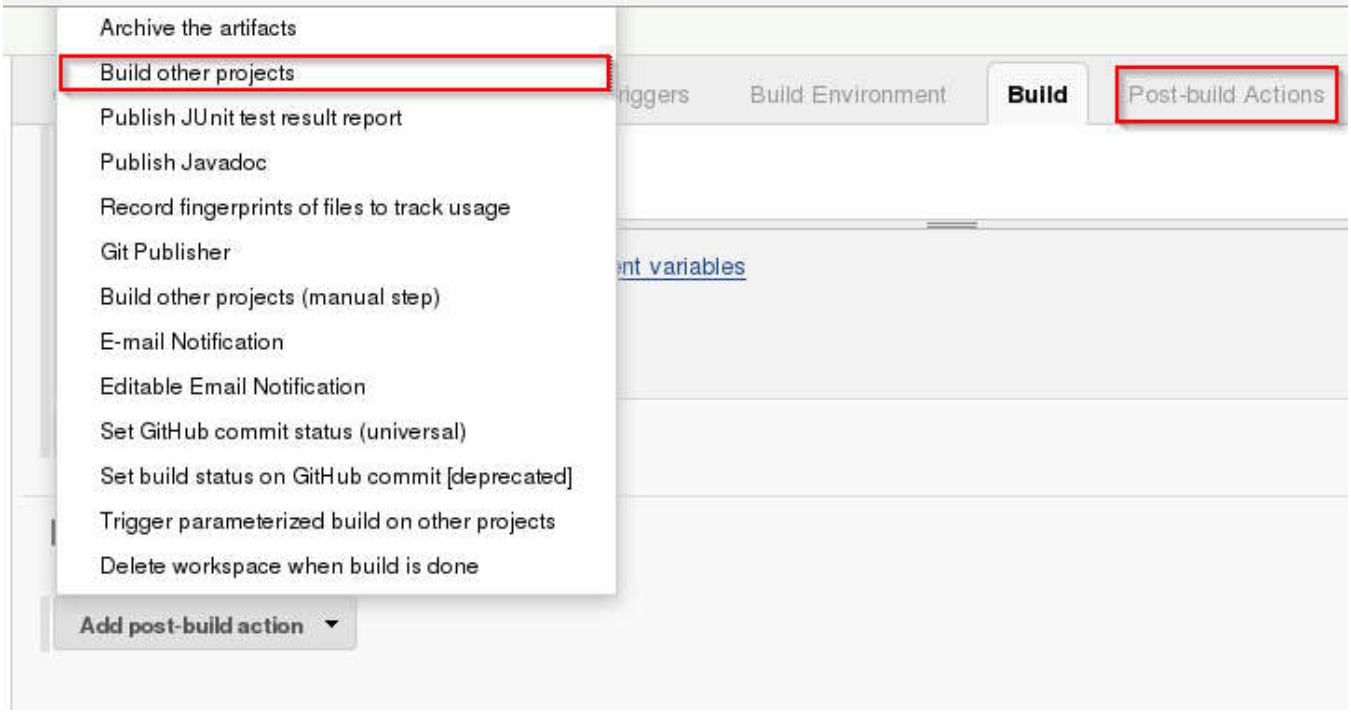
Step 19: Provide the project name to build after Job1 (here is Job2) and then click on **Save.**

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

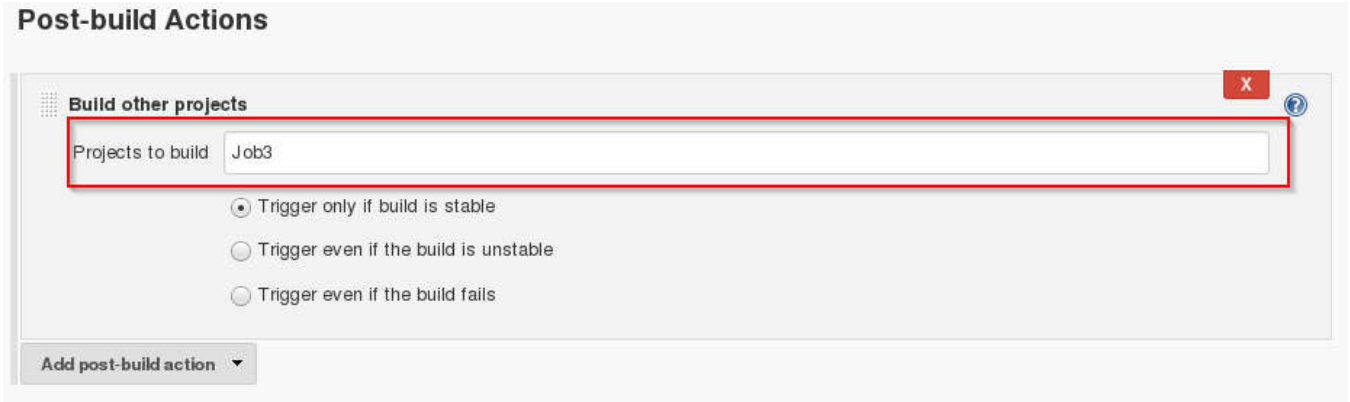




Step 21: Click on **Post-build Actions -> Build other projects.**



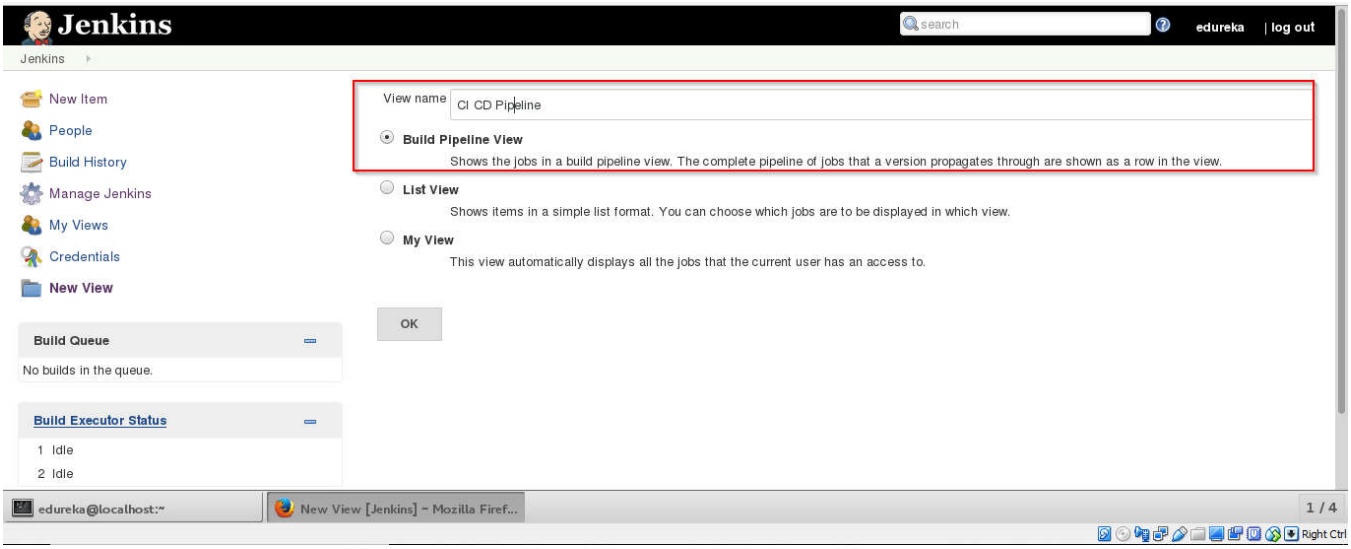
Step 22: Provide the project name to build after Job2 (here is Job3) and then click on **Save.**



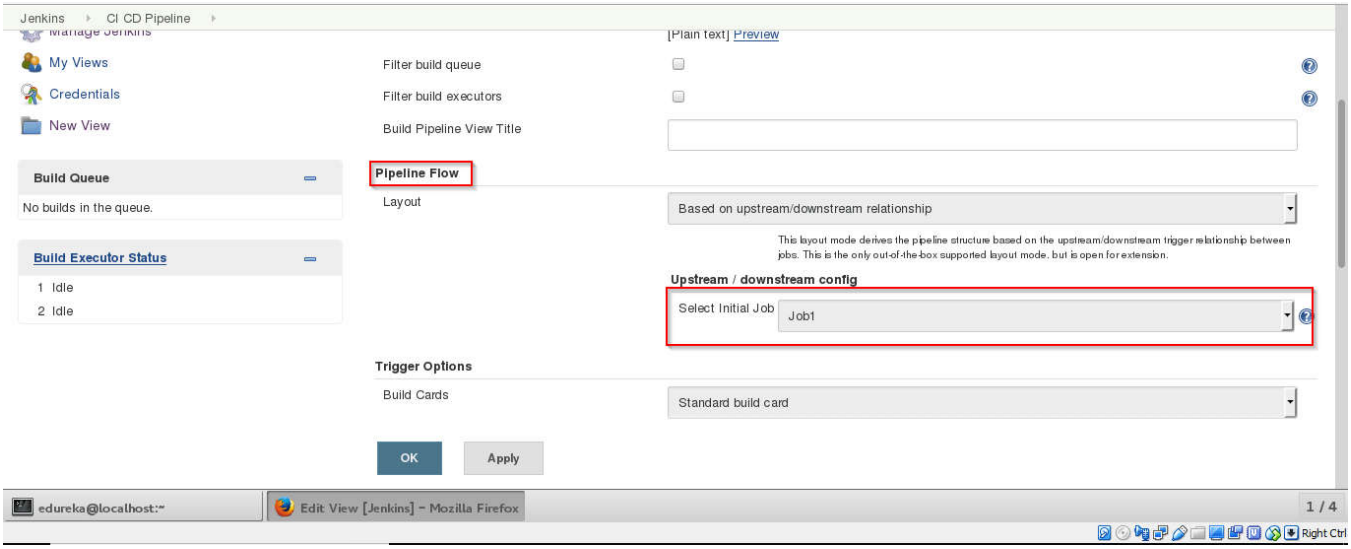
Step 23: Now we will be creating a Pipeline view. Click on '+' sign.

Legend RSS for all RSS for failures RSS for just latest builds

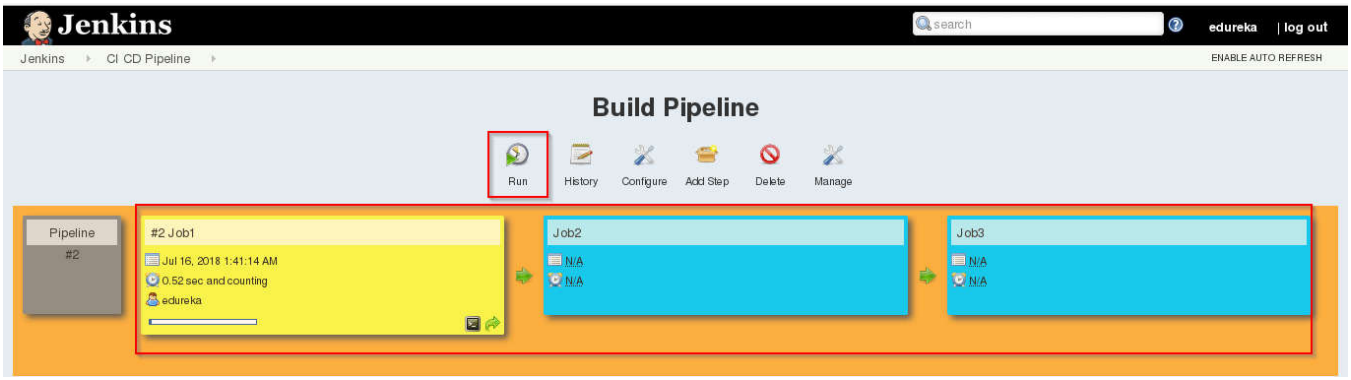
Step 24: Select **Build Pipeline View** and provide the view name (here I have provided CI CD Pipeline).



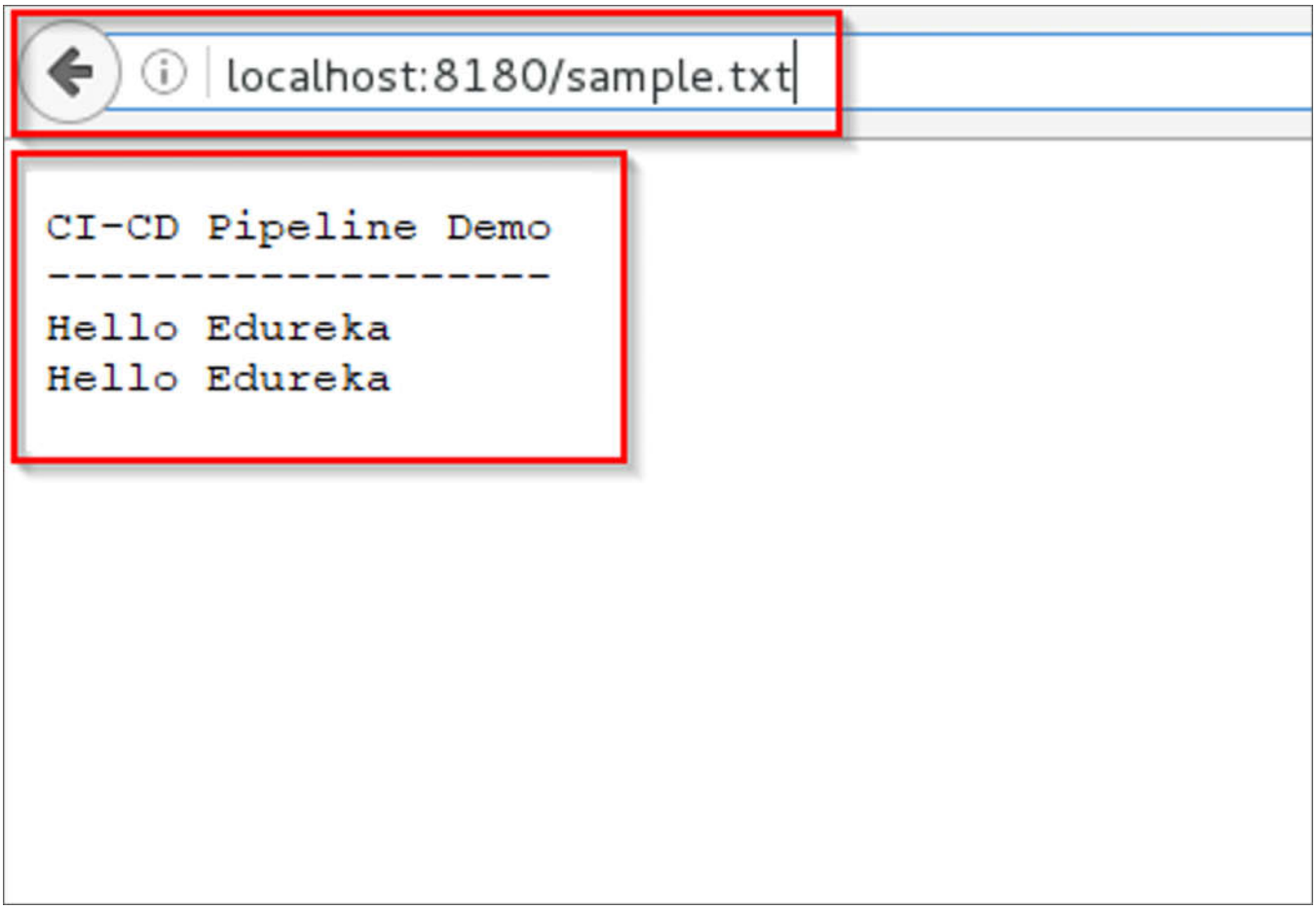
Step 25: Select the **initial Job** (here I have provided Job1) and click on OK.



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 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. If you wish to check out more articles on the market's most trending technologies like Artificial Intelligence, Python, Ethical Hacking, then you can refer to Edureka's official site.