|  |  |  |
| --- | --- | --- |
|  |  |  |

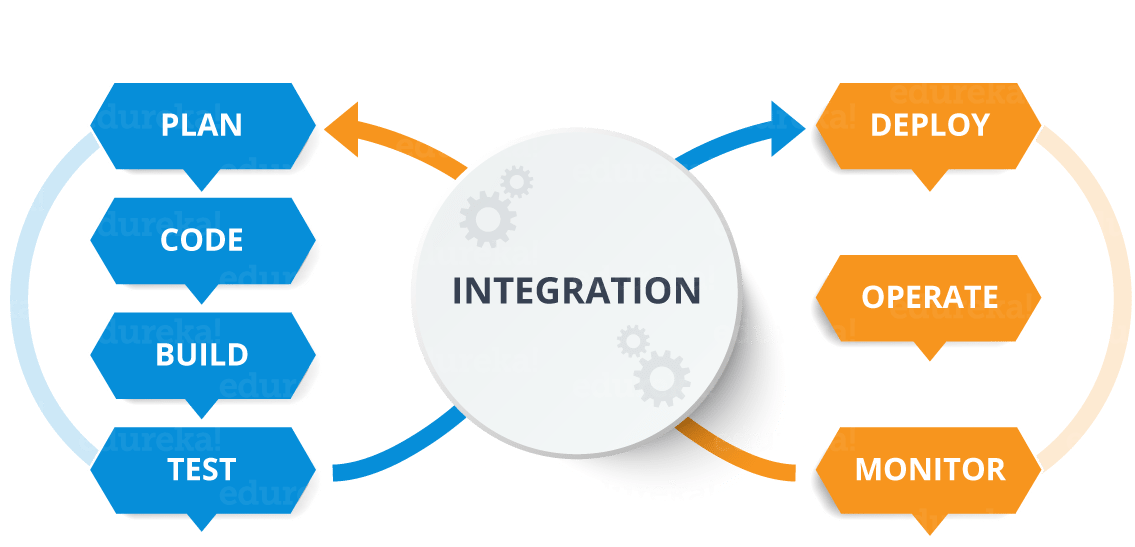
Table of Contents

1. What is CI/CD
2. What is DevOps
3. Ultimate CI Tool and Its Importance in the CI/CD Pipeline
4. What is Jenkins
   1. **- Jenkins Installation**
5. What is Maven
6. How to automate build and deploy in Jenkins (demonstrate CI/CD)
   1. **Java, Maven and Tomcat Setup on Windows Machine**
   2. **Jenkins and GIT HUB Integration**
   3. **Build configuration setting for project**
   4. **Deploy to Container Plugin**
   5. **Deployment configuration setting of project in Tomcat**
7. Jenkins and Sonar Integration for Code Quality

|  |  |
| --- | --- |
| Prepared by | Shubham Chandra |

**What is CI/CD?**

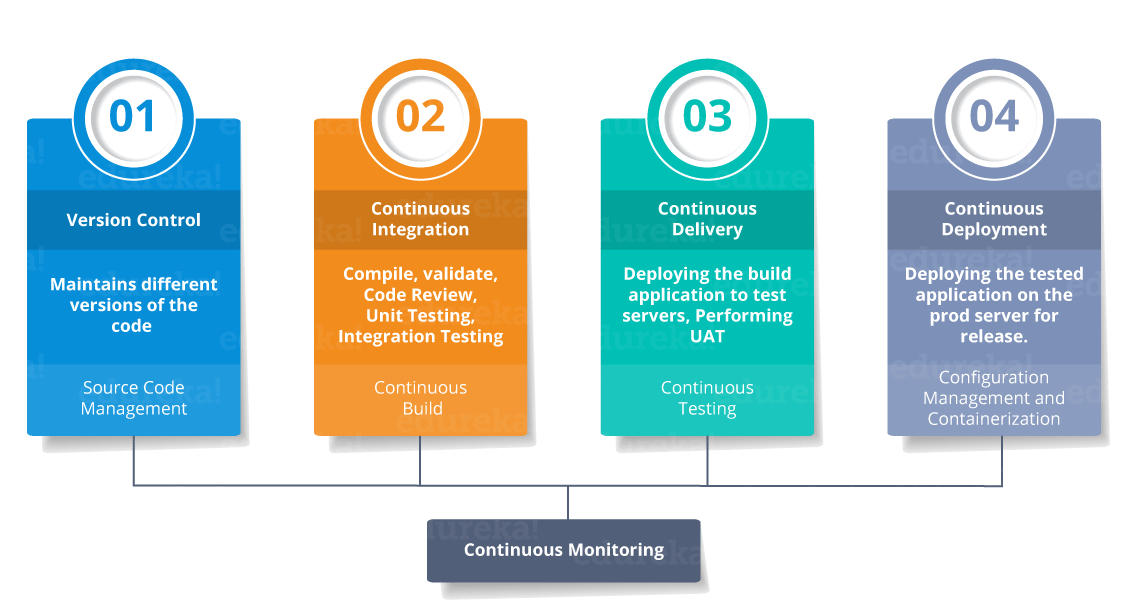
A CI/CD Pipeline implementation, or Continuous Integration/Continuous Deployment, is the backbone of the modern DevOps environment. It bridges the gap between development and operations teams by automating the building, testing, and deployment of applications. In this blog, we will learn what a CI/CD pipeline is and how it works. Before moving onto the CI/CD pipeline, let's start by understanding DevOps.

****

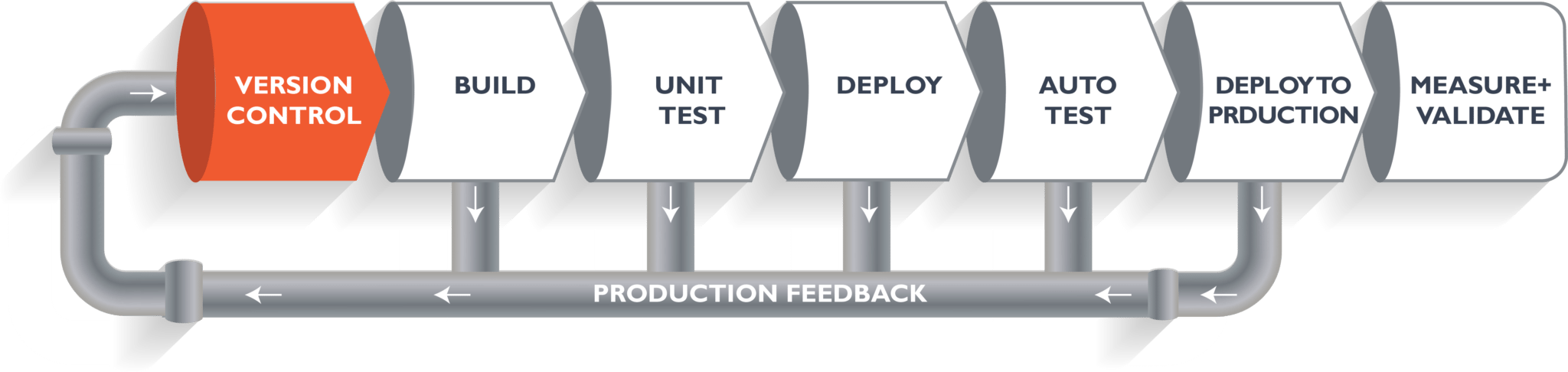
**What is DevOps?**

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 lifecycle. This is the process adopted by all the top companies to develop high-quality software and shorter development lifecycles, resulting in greater customer satisfaction, something that every company wants.

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

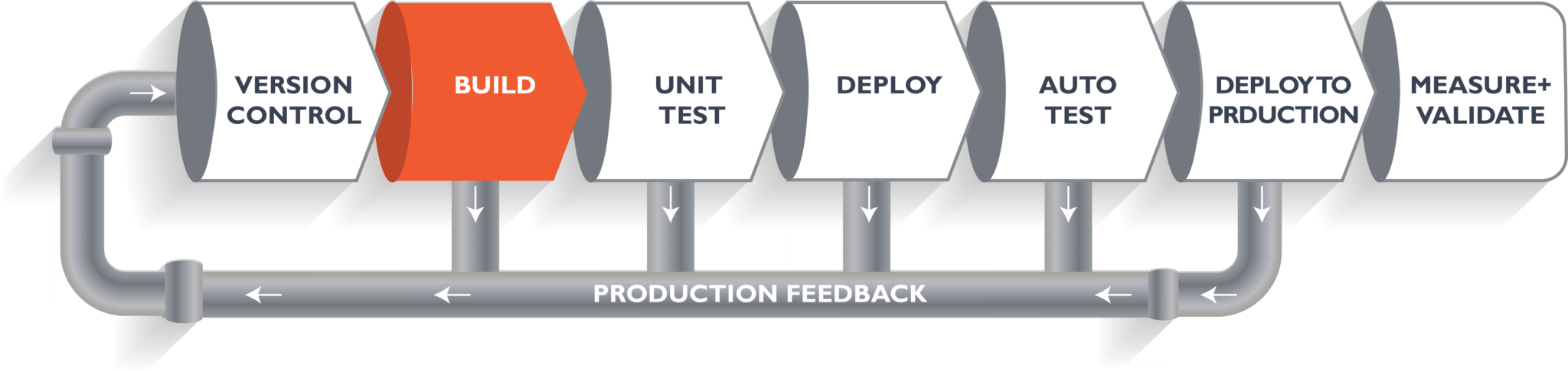


CI stands for Continuous Integration and CD stands for Continuous Delivery/Continuous Deployment. You can think of it as a process similar to a software development lifecycle.  
Let us see how it works.

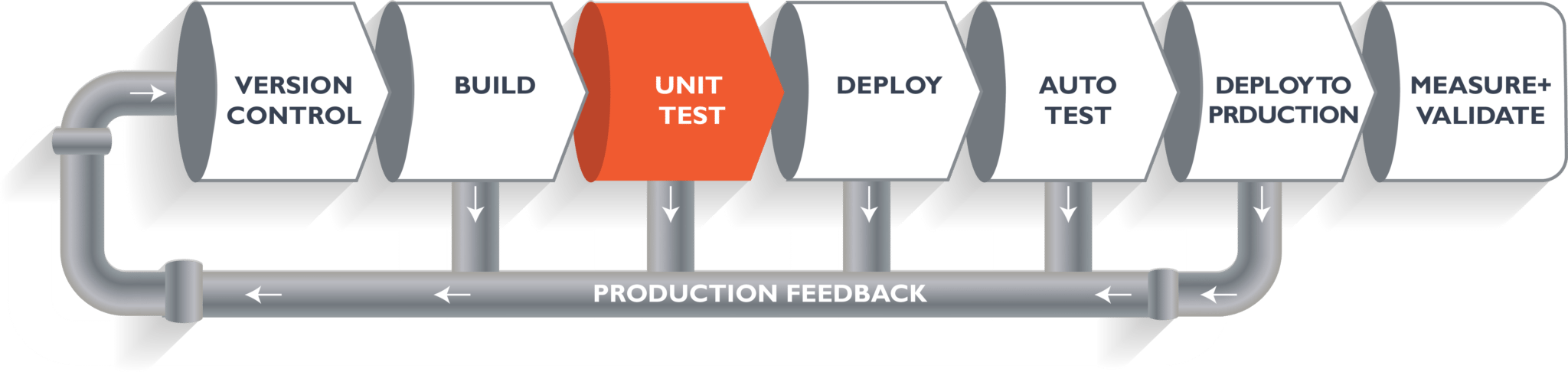


The above pipeline is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production.

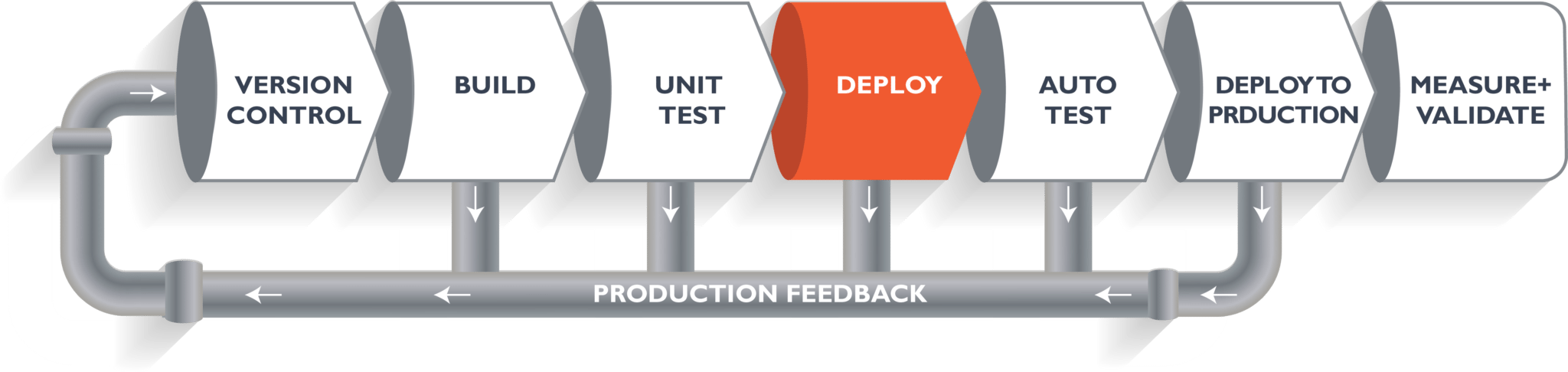
Let's take a scenario of a CI/CD Pipeline. Imagine you're going to build a web application which is going to be deployed on live web servers. You will have a set of developers responsible for writing the code, who will further go on and build the web application. Now, when this code is committed into 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.



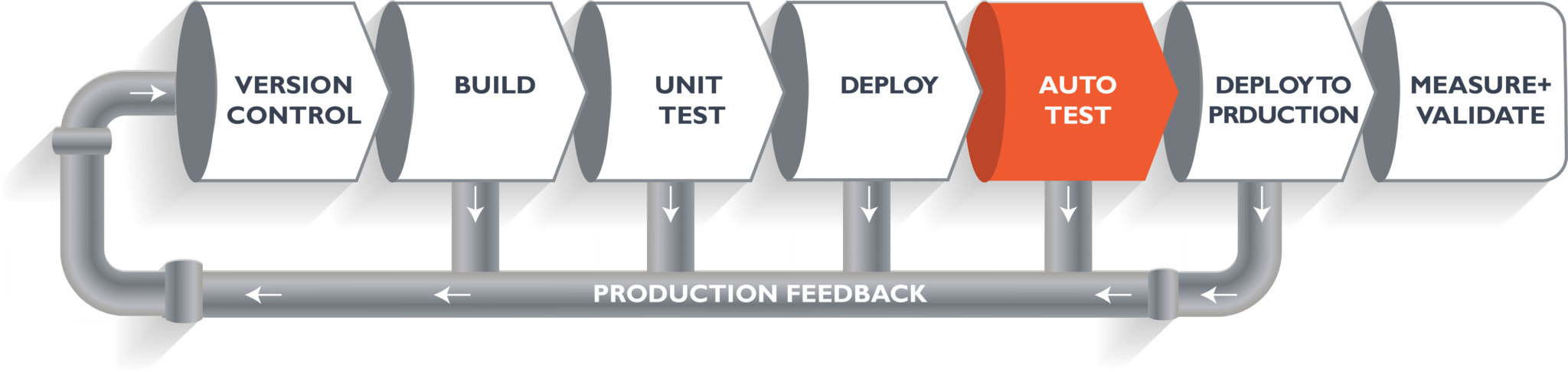
Suppose we have Java code and it needs to be compiled before execution. Through the version control phase, it again goes to the build phase, where it is compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it.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



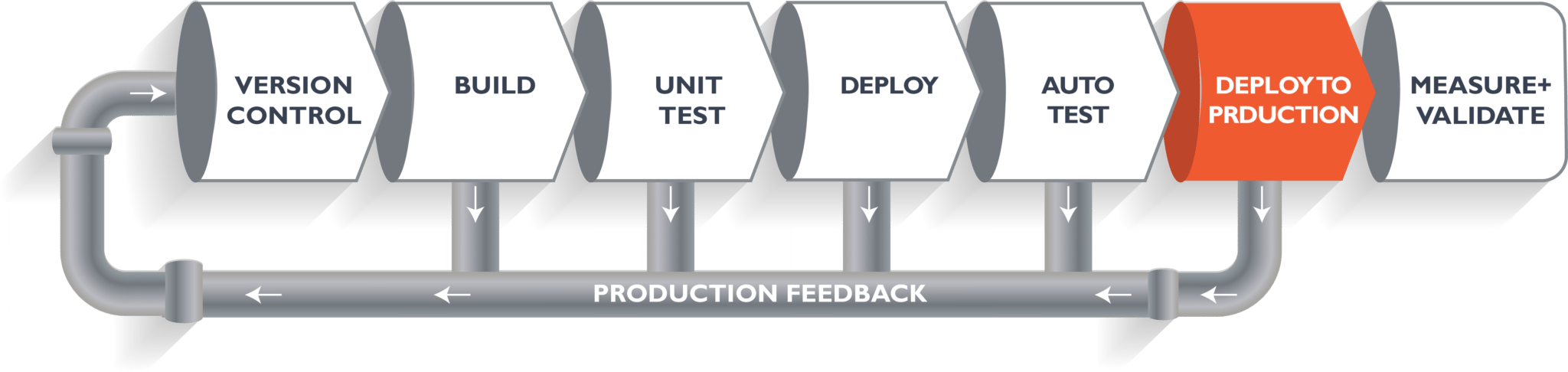
Once the build phase is over, then you move on to the testing phase. In this phase, we have various kinds of testing. One of them is the unit test (where you test the chunk/unit of software or for its sanity test).



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.

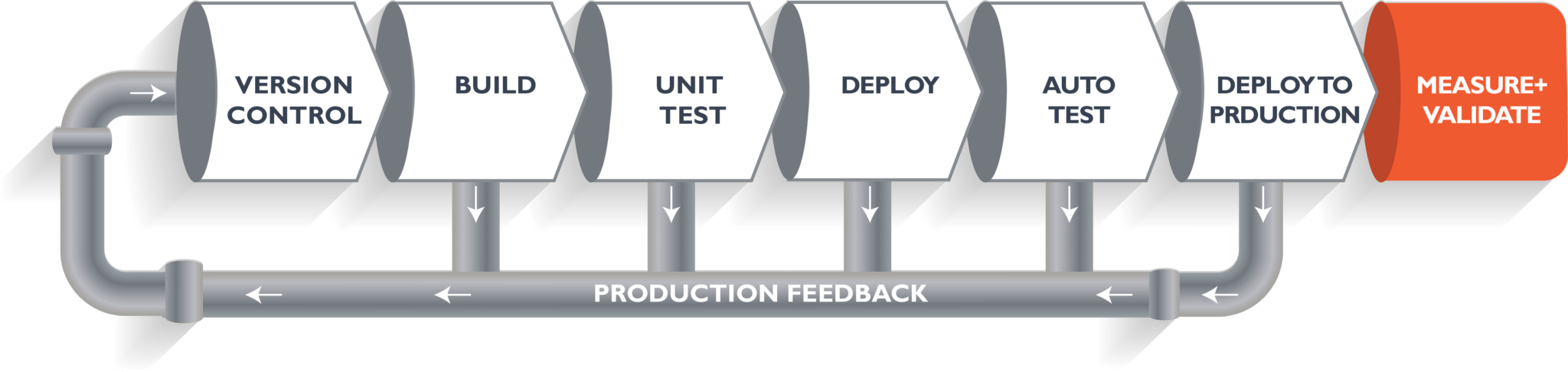


Once the code is deployed successfully, you can run another sanity test. If everything is accepted, then it can be deployed to production.



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.

**Ultimate CI Tool and Its Importance in the 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. We will automate the pipeline in order to make the entire software development lifecycle in DevOps/automated mode. For this, we will need automation tools.



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

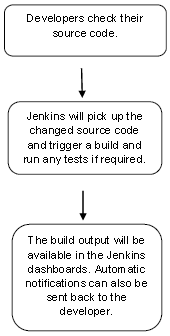
We have a Git repository where the development team will commit the code. Then, Jenkins takes over from there, a front-end tool where you can define your entire job or the task. Our job is to ensure the continuous integration and delivery process for that particular tool or for the particular application.

From Git, Jenkins pulls the code and then Jenkins moves it into the commit phase, where the code is committed from every branch. The build phase is where we compile the code. If it is Java code, we use tools like maven in Jenkins and then compile that code, which 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 **Deployment Tool (uDeploy, Docker etc..).** After a series of unit tests or sanity tests, it moves on to production.

**What is 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. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.



**Jenkins installation->**

* Download Jenkins war from Jenkins website -> [**https://jenkins.io/**](https://jenkins.io/)
* Open the command prompt. From the command prompt, browse to the directory where the jenkins.war file is present. Run command -> **D:\>Java –jar Jenkins.war**
* Once Jenkins is up and running, one can access Jenkins from the link − [**http://localhost:8080**](http://localhost:8080)
* Below message can see in command prompt screen when Jenkins up.

*INFO: Completed initialization*

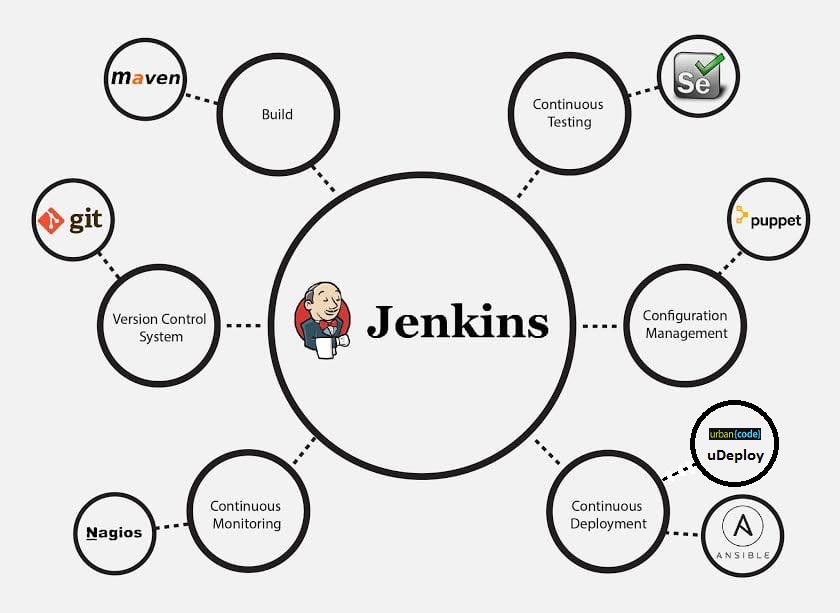
*Jul 06, 2019 11:43:04 AM hudson.WebAppMain$3 run*

*INFO: Jenkins is fully up and running*

**NOTE:** To start Jenkins on any other port can use below command

**java -jar jenkins.war --httpPort=9090**

* Using Jenkins can integrate multiple tools and automate build, code quality check, unit testing, and deployment in one click, will go step by step and complete all points.

****

**What is Maven?**

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation.

Maven does following things for building an application.

* Generates source code
* Generates documentation from source code
* Compiles source code
* Packages compiled code into JAR/WAR/ZIP file
* Installs the packaged code in local repository, server repository, or central repository

**How to automate build and deploy in Jenkins**

Here we are going to build java maven project and deploy it on tomcat apache server. Below plugins and setup required for automating jobs via Jenkins.

1. Java, Maven and Tomcat Setup on Machine.
2. Jenkins Integration with GIT (GIT SCM Plugin).
3. Build configuration setting for project.
4. Deploy to Container Plugin.
5. Deployment configuration setting of project in Tomcat.
6. **Java, Maven and Tomcat Setup on Machine**
   1. **Java**
7. Download and install latest java on machine and set JAVA\_HOME and PATH environment and System variables for jdk as below.

*JAVA\_HOME = C:\Program Files\Java\jdk1.8.0\_121*

*PATH = %JAVA\_HOME%\bin*

1. Validate if java path set properly by using below cmd in command prompt.

*java –version*

* 1. **Maven**

1. Download and install Maven on machine and set M2\_HOME and PATH

environment and System variables for jdk as below.

*M2\_HOME = C:\Program Files\Maven*

*PATH = %M2\_HOME%\bin*

1. Validate if maven path set properly by using below cmd in command prompt.

*mvn*

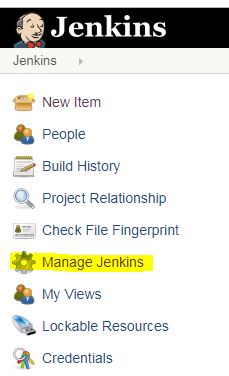
* 1. **Tomcat**
     1. Now have to start Tomcat before we start have to make sure tomcat should run on different port other than Jenkins one. To make sure same change port in *server.xml* file available (Connector port) inside *conf* folder.



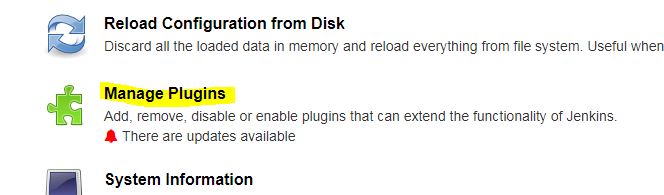
* + 1. Start tomcat 'catalina.bat start' from bin folder.
    2. Now create tomcat users in tomcat-user.xml file inside conf folder.
    3. Start the Tomcat Server again and login in by clicking on Server Status.

1. **Jenkins and GIT HUB Integration (GIT SCM plugin)**

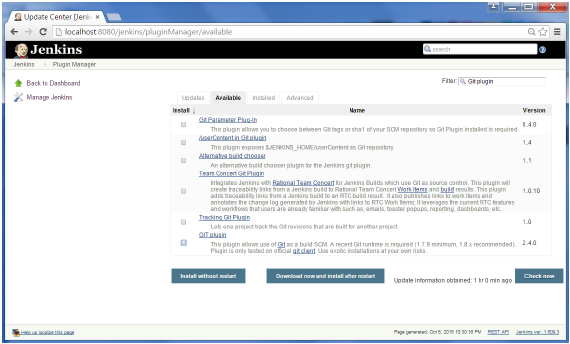
* In your Jenkins Dashboard (Home screen), click the Manage Jenkins option on the left hand side.



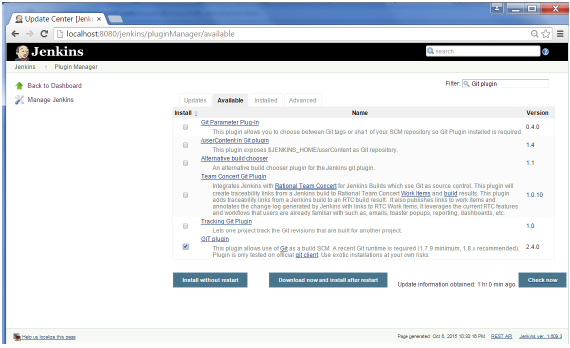
* In the next screen, click the ‘Manage Plugins’ option.



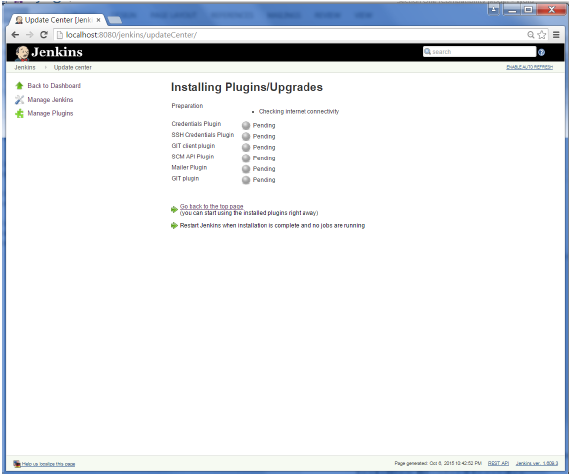
* In the next screen, click the Available tab. This tab will give a list of plugins which are available for downloading. In the ‘Filter’ tab type ‘Git plugin’



* The list will then be filtered. Check the Git Plugin option and click on the button ‘Install without restart’

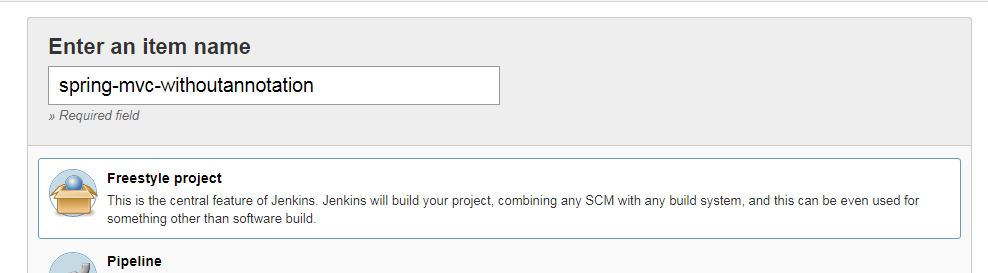


* The installation will then begin and below screen will show the status of the download.



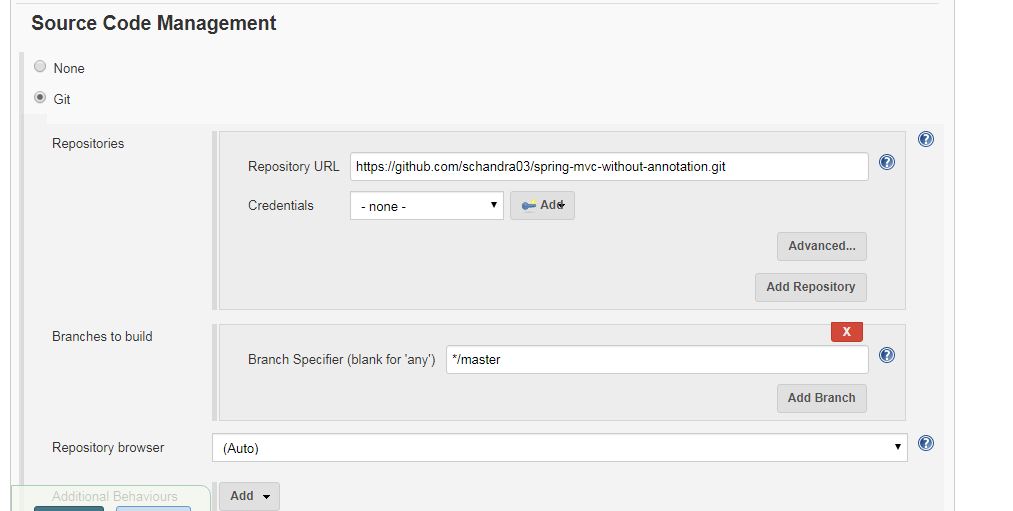
* Once Jenkins is restarted, Git will be available as an option in configuring jobs. To verify, click on New Item in the menu options for Jenkins. Then enter a name for a job, in the following case, the name entered is “spring-mvc-without-annotation”. Select ‘Freestyle project’ as the item type. Click the Ok button.

**Note:** We have one spring project which will build and deploy on Tomcat Server.

 🡪 

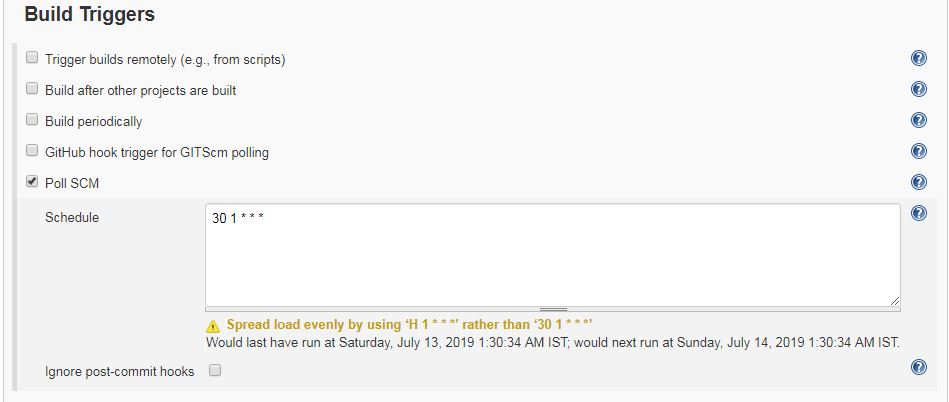
* In the next screen, if you browse to tab ‘*Source code Management*’ section, you will now see ‘Git’ as an option. Repository URL will be GIT link of project and branch will be *master*.

**Note:** Below is GIT link of project, will use same in our configuration.<https://github.com/schandra03/spring-mvc-without-annotation>



* If you want your Jenkins will trigger build for each and every GIT commit or any specific interval of time then can use below *Poll SCM* option.

Eg: Below Jenkins will do build every day 1:30 AM IST.



1. **Build configuration setting for project.**

* Under Build option have to provide mvn clean package to build the project like below.

****

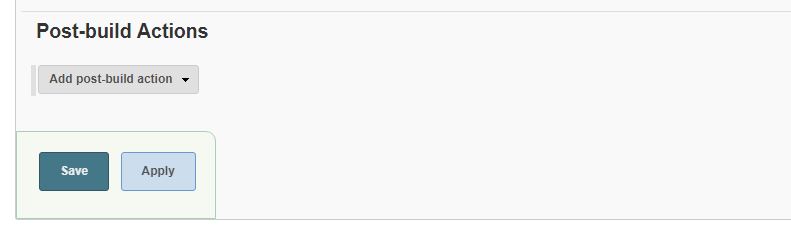
1. **Deploy to Container Plugin**

* Like Jenkins plugin have to install container plugin.



1. **Deployment configuration setting of project in Tomcat**

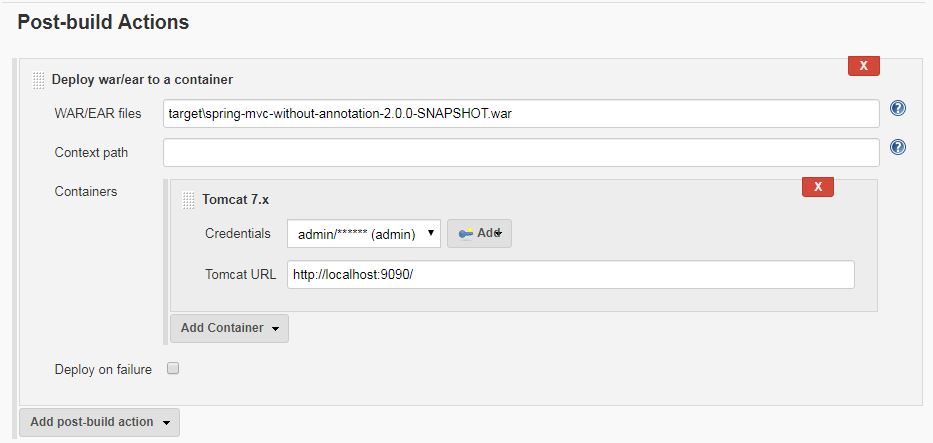
* Once container plugin install go to your Jenkins Job (spring-mvc-without-annotation) and go to option *Post-build Actions.*



* Click on *Add post-build action* and select *Deploy war/ear to container.*  In WAR/EAR files have to provide the WAR file name with path.

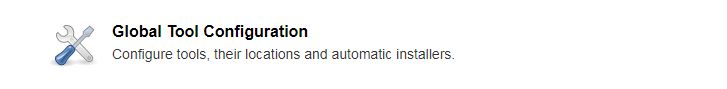
**Note:** By default Jenkins will package source code in *target* folder of the Jenkins Workspace. In my case ear file present in *C:\Users\SHUBH\.jenkins\workspace\spring-mvc-without-annotation\target folder.*

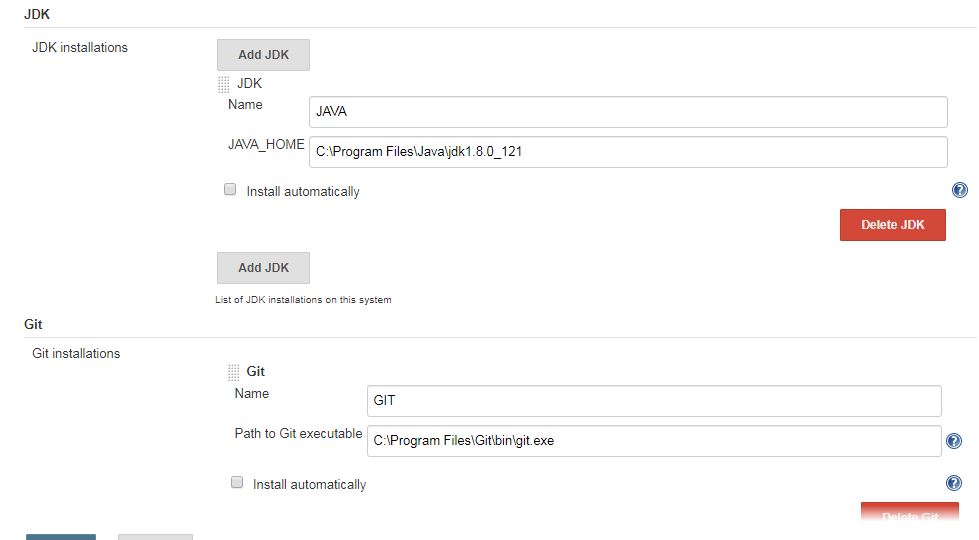
* In *Containers* have to provide the tomcat user credentials and complete URI with hostname and port as below.

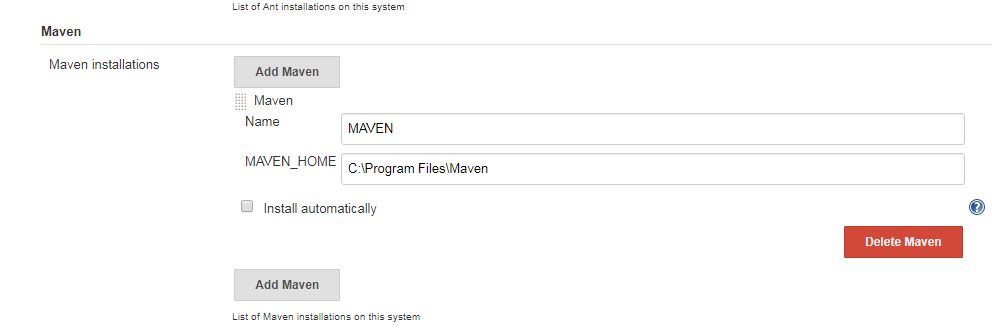


* In 'Global Tool Configuration' option under *Manage Jenkins* need to provide path for below installations.

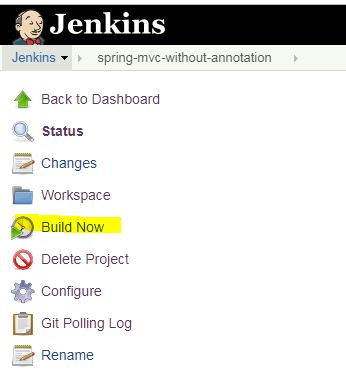
1. JDK
2. Maven
3. GIT







* After that trigger the build, it will deploy the war file in tomcat.

****

* Once deployment is done then can validate application and confirm if up and running using below URL.

<http://localhost:9090/spring-mvc-without-annotation-2.0.0-SNAPSHOT/userRegistration.htm>

**Jenkins and Sonar Integration:** For Code Quality

* + - 1. Download compatible version of SonarQube and sonar-scanner.
      2. Start the SonarQube Server.

2.1 Go to bin folder of downloaded SonarQube server in cmd prompt and execute *StartSonar.bat* cmd like below.

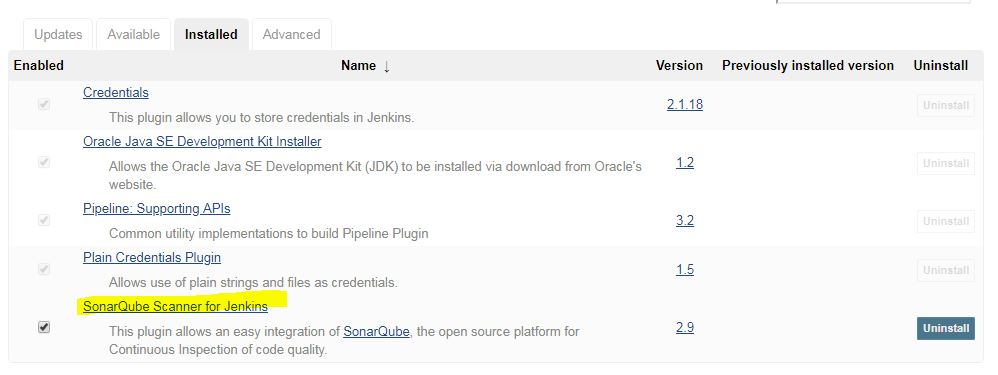
2.2 Once “SonarQube is up” then hit below localHostURL to valid if Sonar is up or not.

<http://localhost:9000/projects> (assuming running on default port 9000)

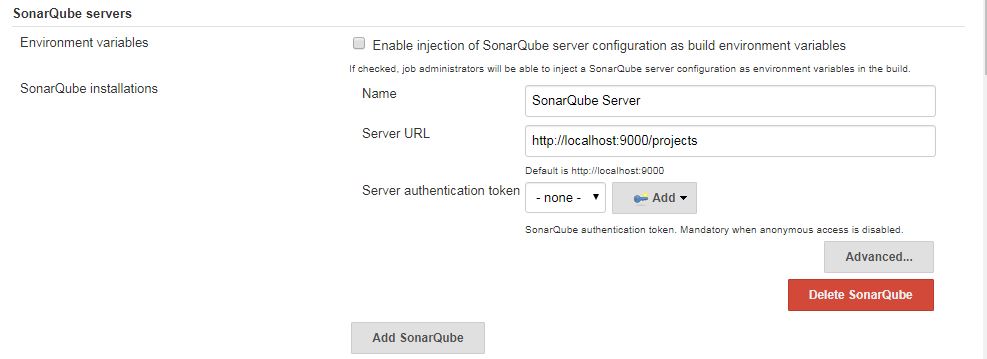


**Note:** If you want to stop SonarQube Server then Go to same cmd prompt in which it is running and press Ctrl+C.

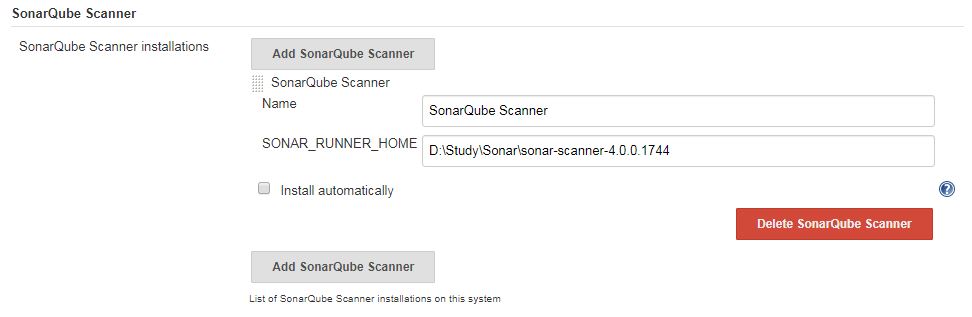
* + - 1. Now go to Jenkins Dashboard Click on “Manage Jenkins” - > “Manage Plugins”. In Available plugin search for “Sonar Qube Scanner For Jenkins” plugin. Install that plugin.



* + - 1. Now go to “Configure System” under “Manage Jenkins” and Add Sonar Qube Server host and port like below.



* + - 1. Again to “Manage Jenkins” -> “Global Tools Configurations” . Add SonarQube Scanner Installation paths like below (which we download in point first).



* + - 1. Now Apply and Save all setting and go to your project. In my case Jenkins project name is “spring-mvc-without-annotation”.
      2. Under Jenkins project click on “Configure” and provide below cmd in “Build” section.

mvn clean package sonar:sonar



* + - 1. Once above tasks done save all settings and trigger Jenkins Build, below cmd will create package and also run Sonar Runner to check the code quality and send the report to SonarQube server.

**mvn clean package sonar:sonar**

* + - 1. Go to Sonar Quber Server Dashboard using link <http://localhost:9000/projects> and your scanned project will show there.

