**Jenkins**

**What is Jenkins and why we use it?**

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. 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.

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

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example Git, Maven 2 project, Amazon EC2, HTML publisher etc.

### Advantages of Jenkins include:

* It is an open-source tool with great community support.
* It is easy to install.
* It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
* It is free of cost.
* It is built with Java and hence, it is portable to all the major platforms.

**Before and After Jenkins**

|  |  |
| --- | --- |
| **Before Jenkins** | **After Jenkins** |
| The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time-consuming, which in turn slows the software delivery process. | Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases. |
| Developers have to wait for test results | Developers know the test result of every commit made in the source code on the run. |
| The whole process is manual | You only need to commit changes to the source code and Jenkins will automate the rest of the process for you. |

**Alternatives Tools of jenkins**

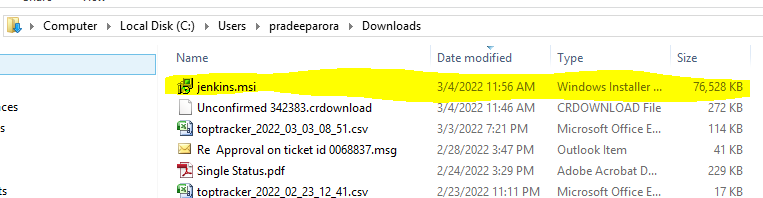
* Atlassian Bamboo
* JetBrains TeamCity
* AWS CodePipeline
* Azure DevOps Server (formerly Microsoft Team Foundation Server)
* IBM Urbancode.
* GitLab CI.

**Jenkins Installation**

Go to jenkins official site

Link: https://www.jenkins.io/download/#downloading-jenkins

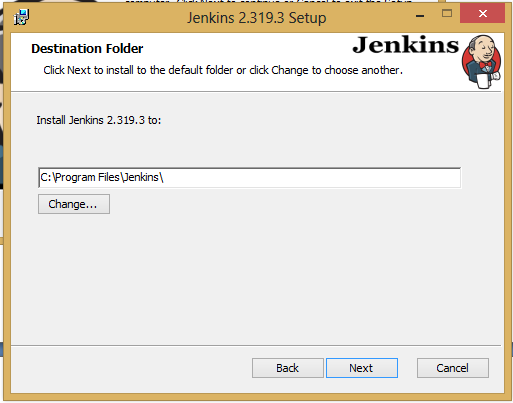
Check in download folder

****

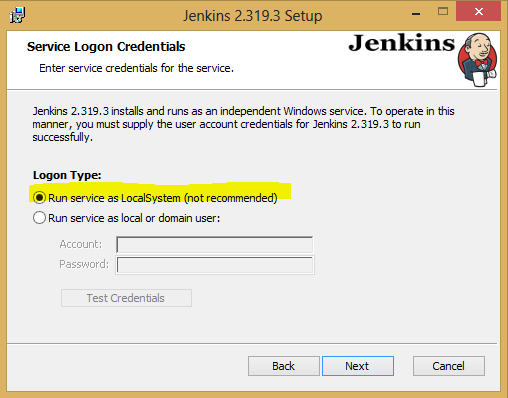
Right click on Jenkins.msi file and run as administrator



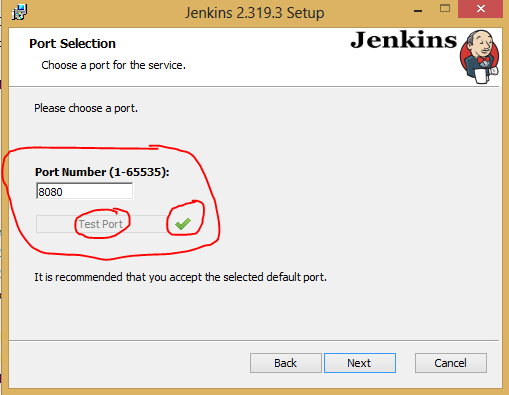
Click on Next.



Select Jenkins Install location by default location C:\Program Files\Jenkins\ if we want to change default location.

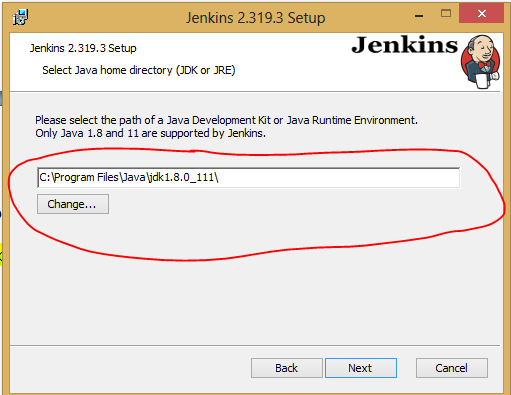


Select Run Service as LocalSystem.

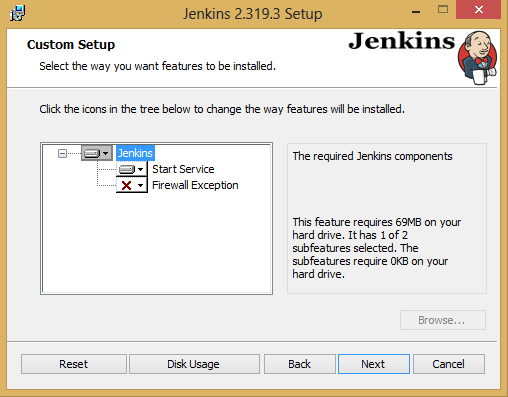


Enter Port Number and Test Port

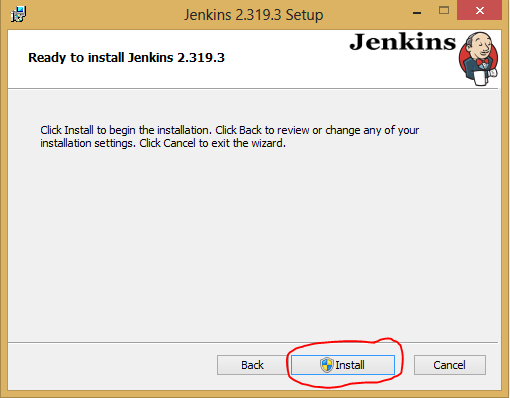
Default port of Jenkins : 8080



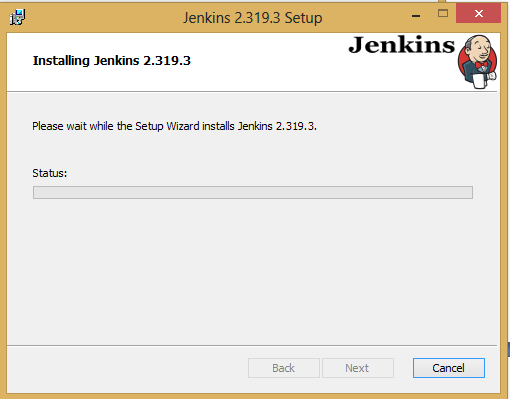
Select JDK location or change of you want other JDK installed version.



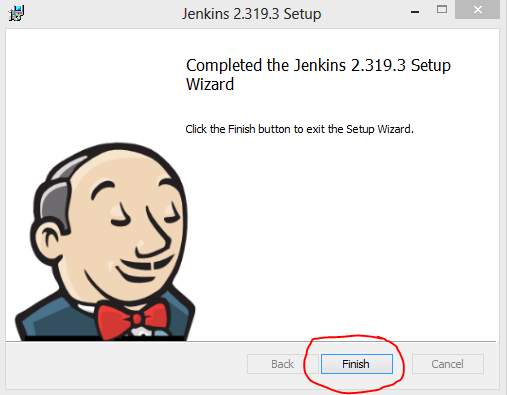
Click on Next



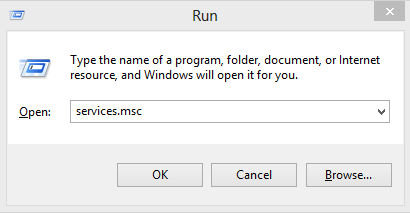
Click on Install.



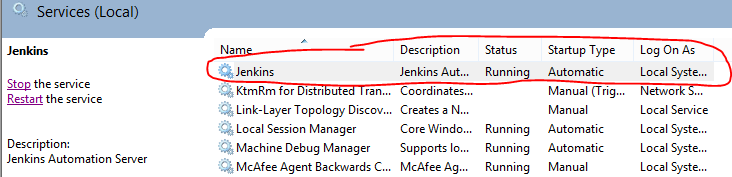
Wait for installation.



Click on Finish.



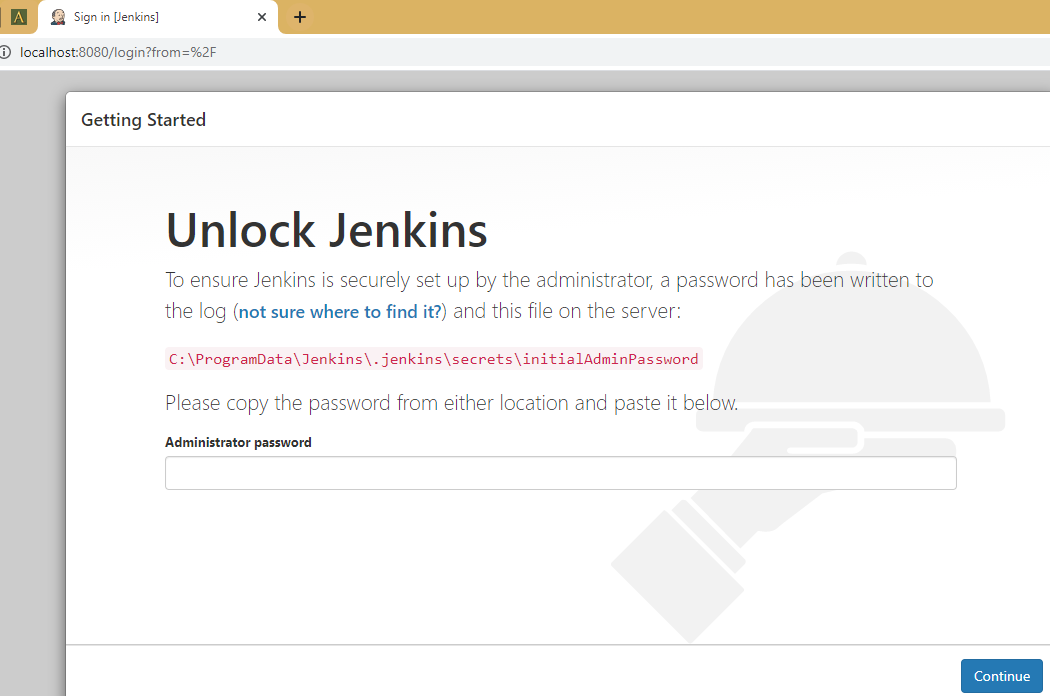
Open Services.msc



Check Jenkins service status that running or not.

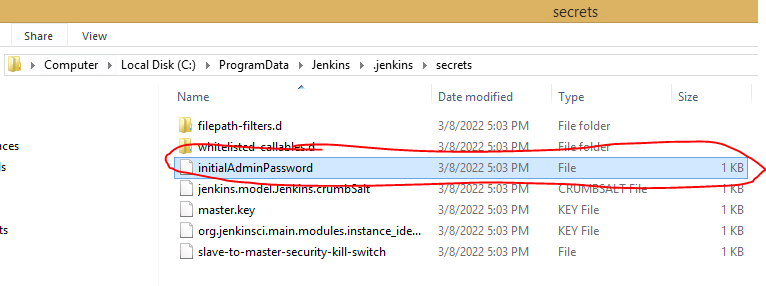
Open in browser.

http://localhost:8080/



Unlock Jenkins and Setup administrator password.

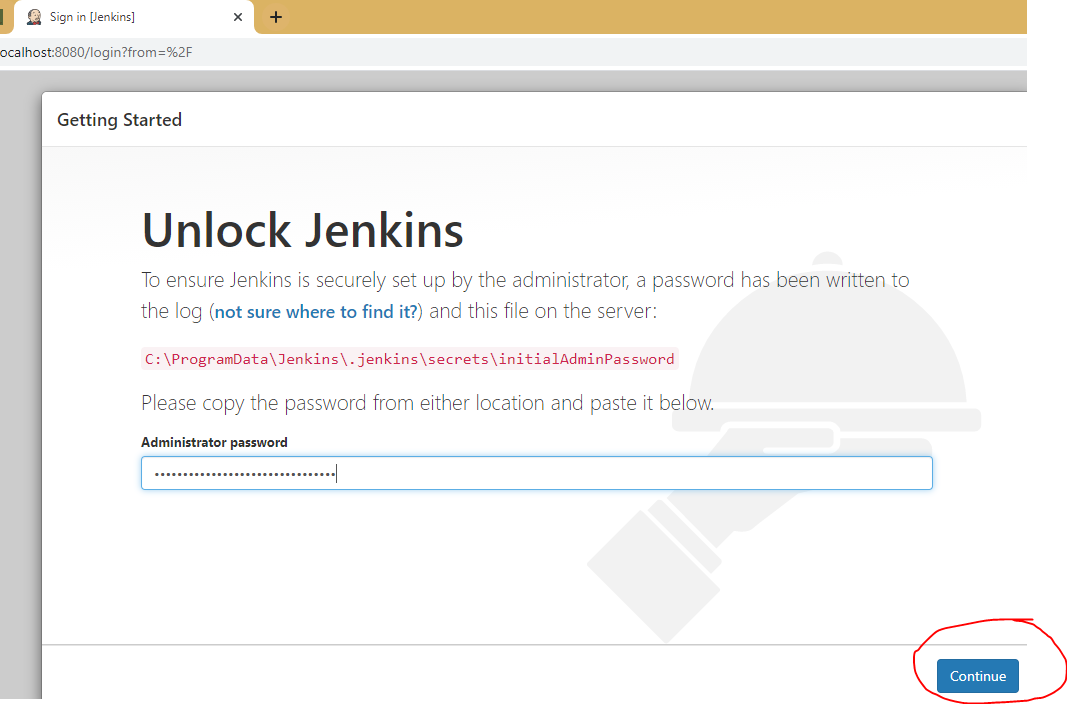
Go to Location : **C:\ProgramData\Jenkins\.jenkins\secrets** and Open **initialAdminPassword file**



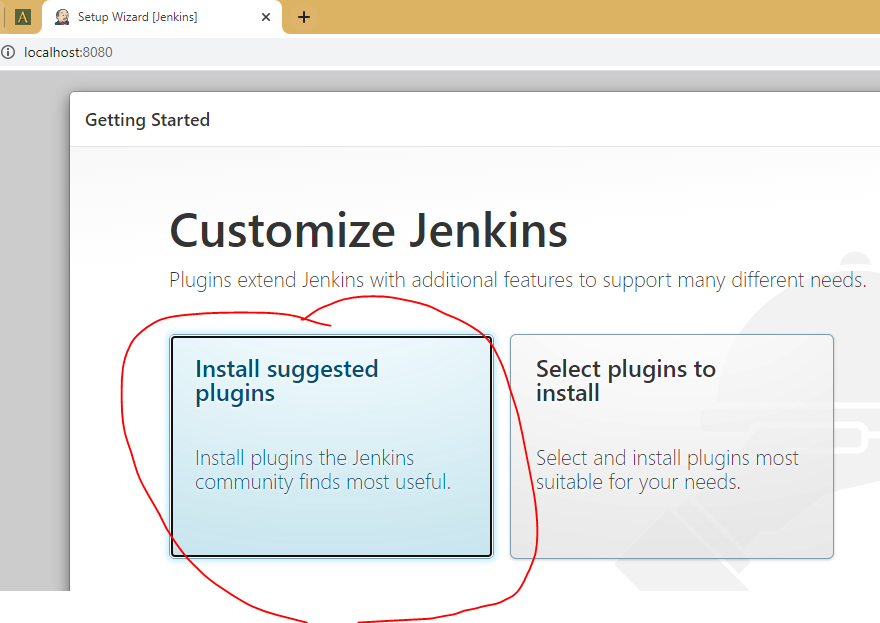
Open File



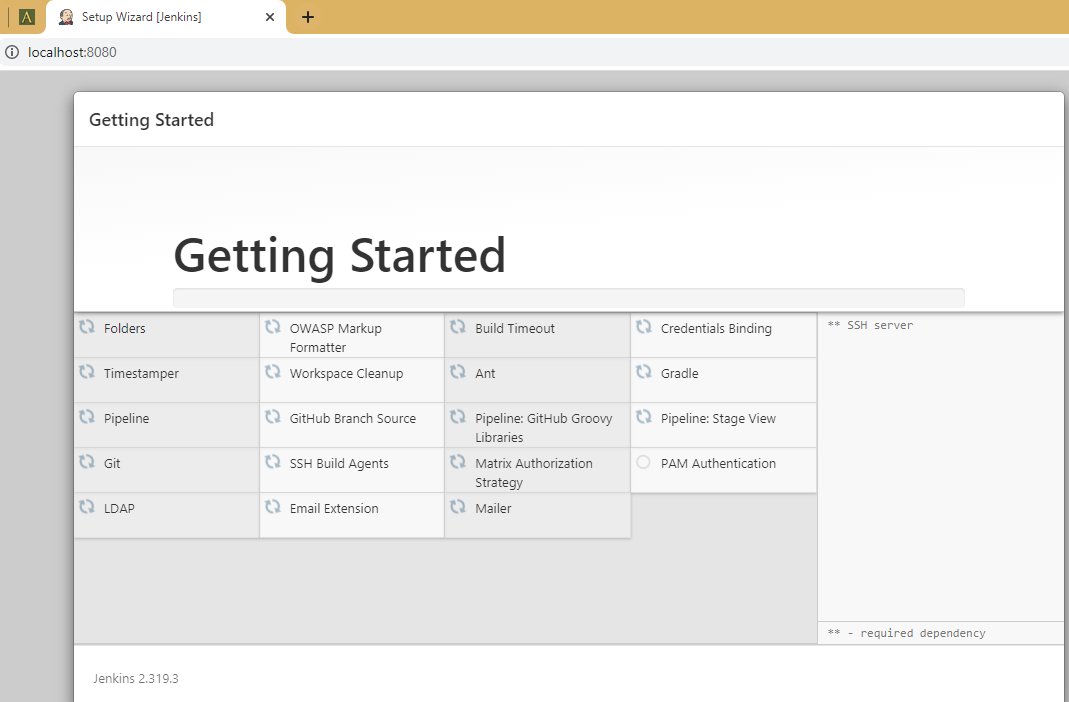
Copy this content and paste in Jenkins admin password.



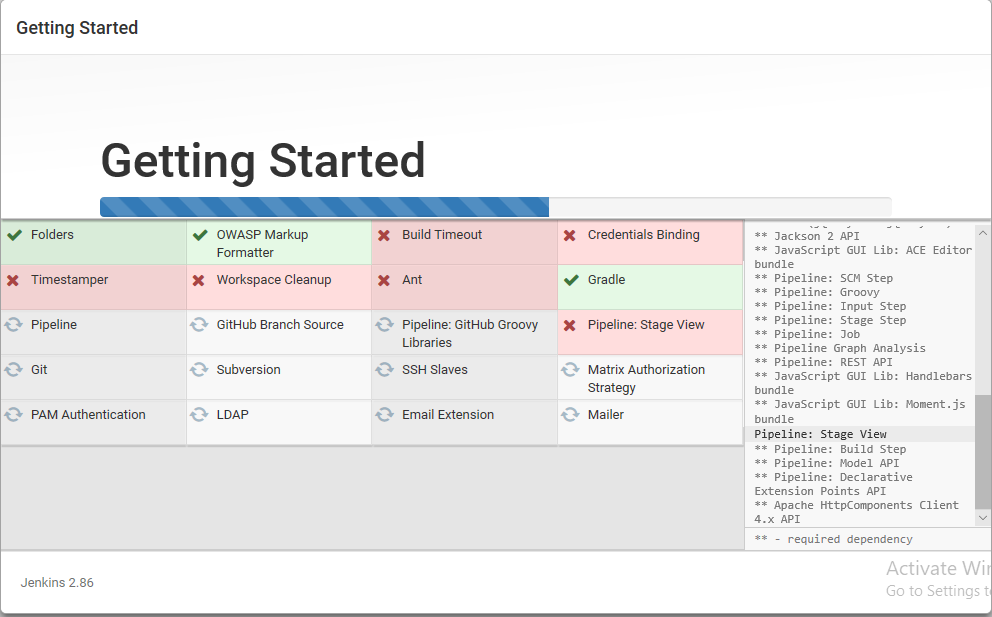
Enter Administrator password and click on continue.

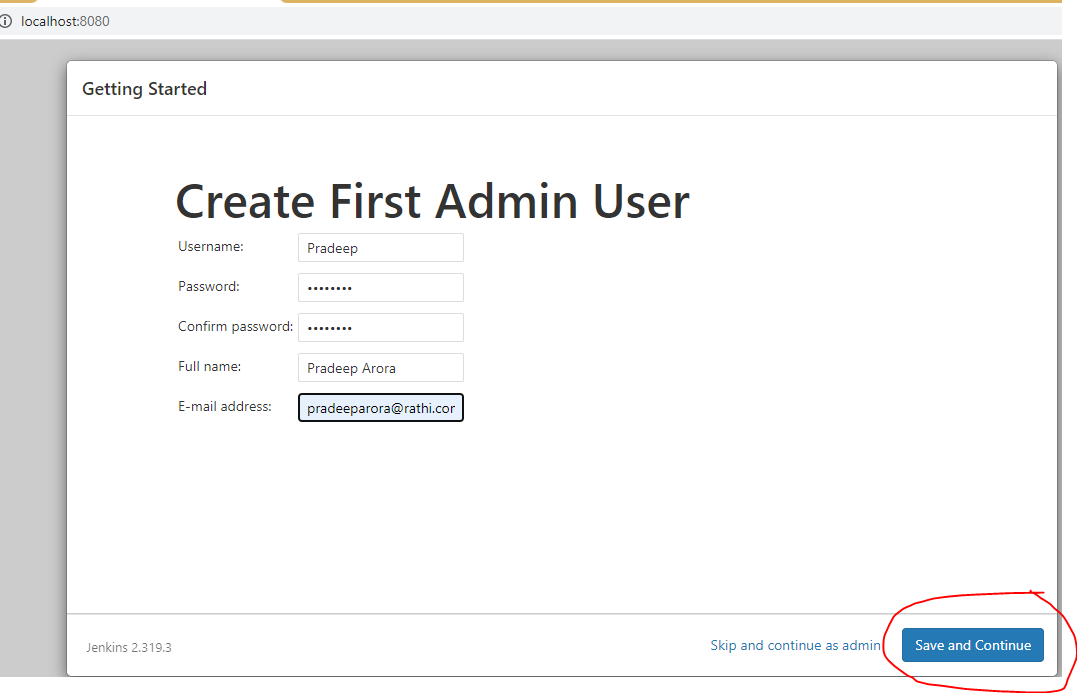


Click on Install Suggested Plugging for all default Jenkins Plug-in.

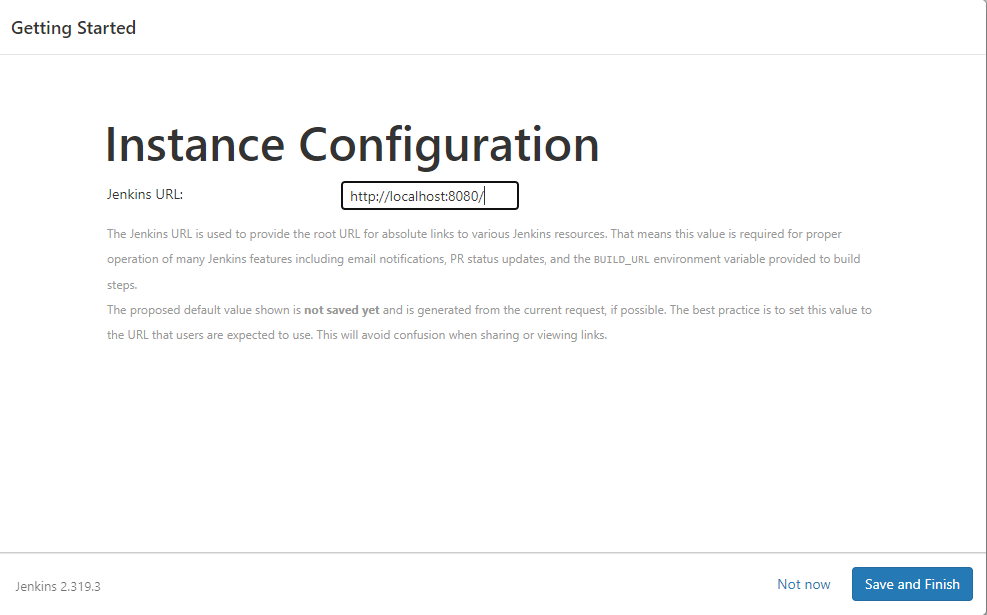


It will take some time to install Plug-in.

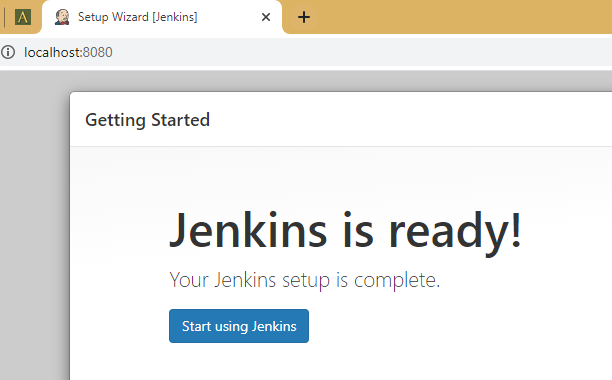




-> create First admin user and click on save and continue.

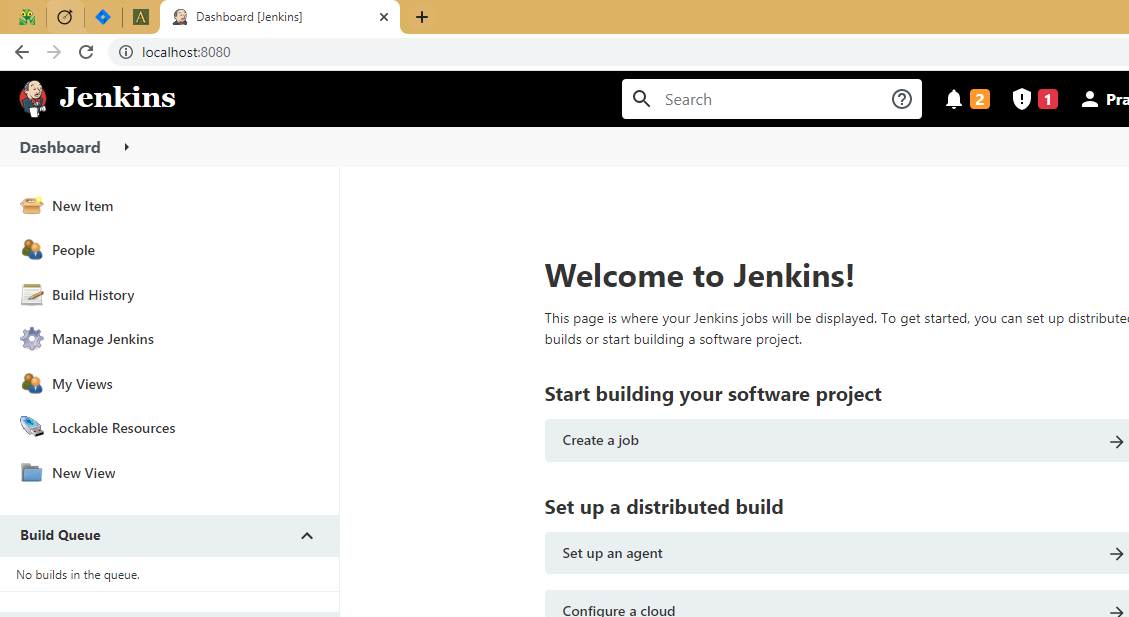


-> Set URL and Click on Save and Finish.



-> Now Jenkins is ready to use.

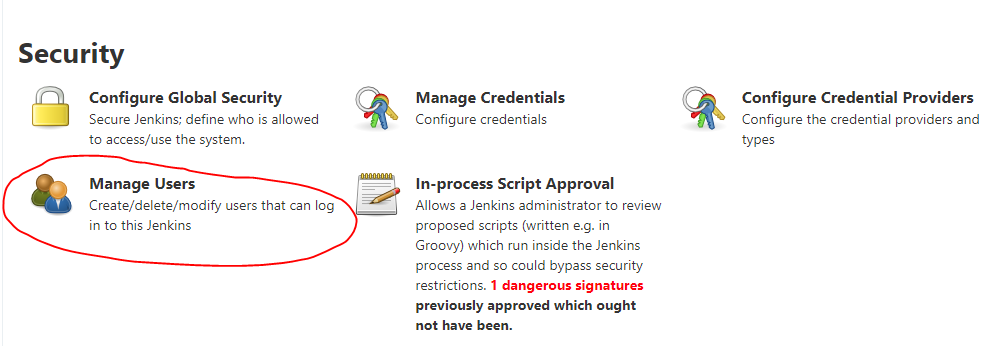
-> Click on Start using Jenkins.



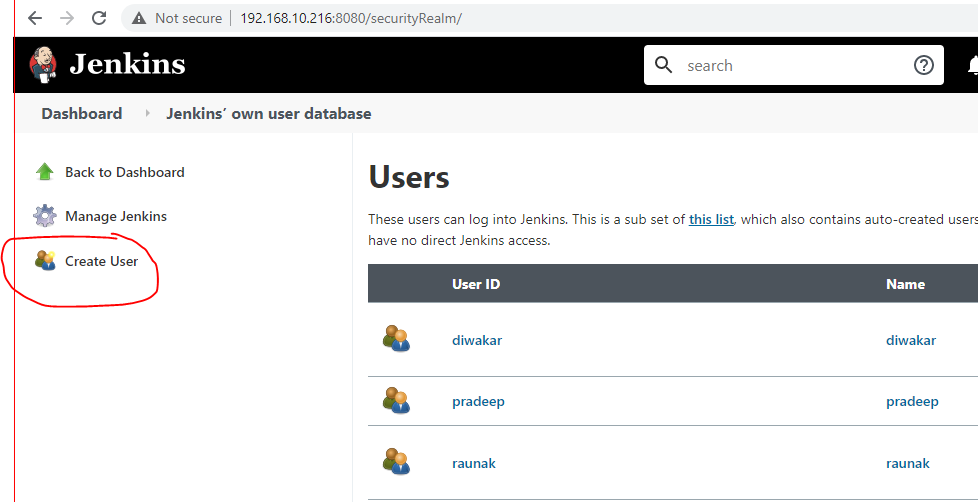
**Create New User**

Go to Jenkins Dashboard

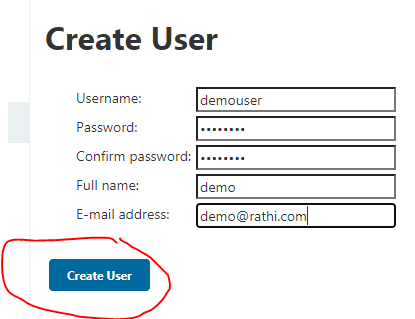
Go to Manage Jenkins > Security >ManageUser> Create User



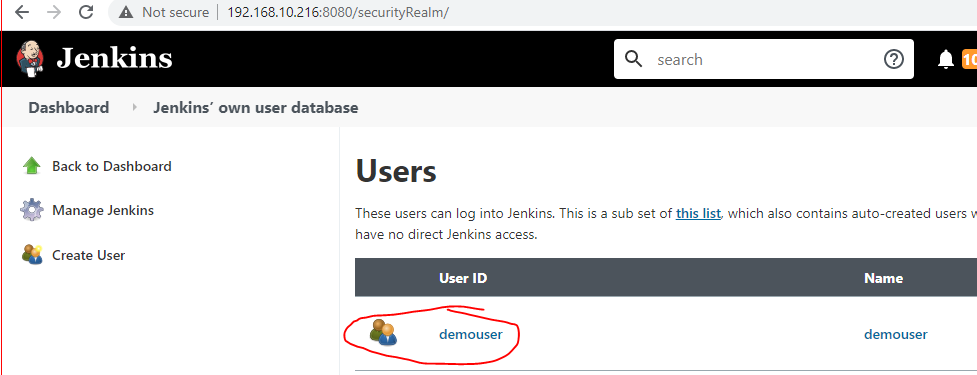
Click on Manage Users



Click on Create User



Enter Information of username, password, FullName and Email and click on create user.

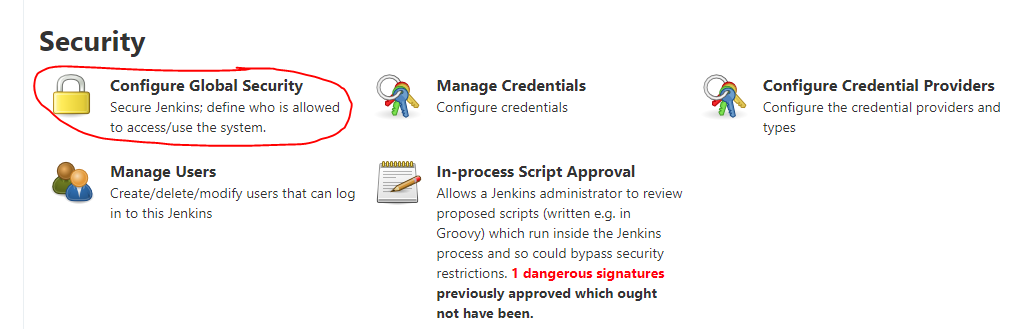


User is created Successfully.

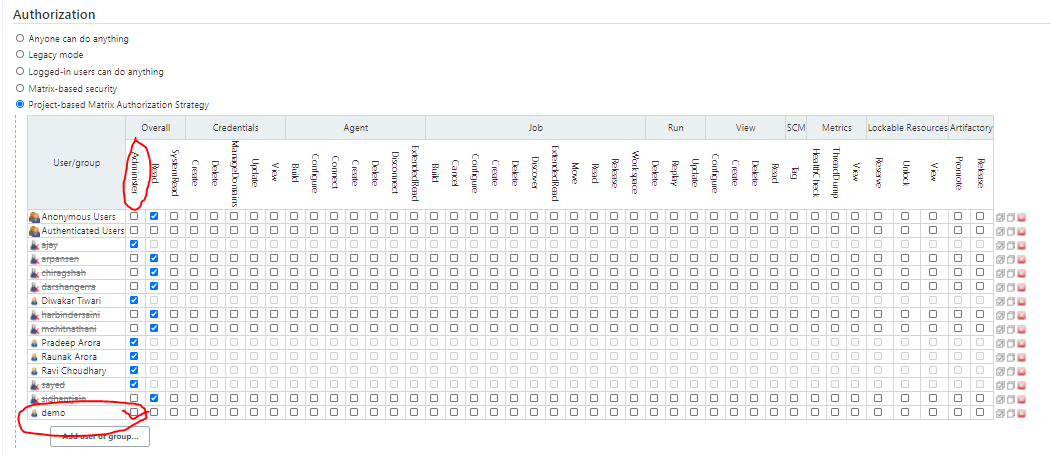
**Provide Admin Access to User**

Go to Jenkins Dashboard

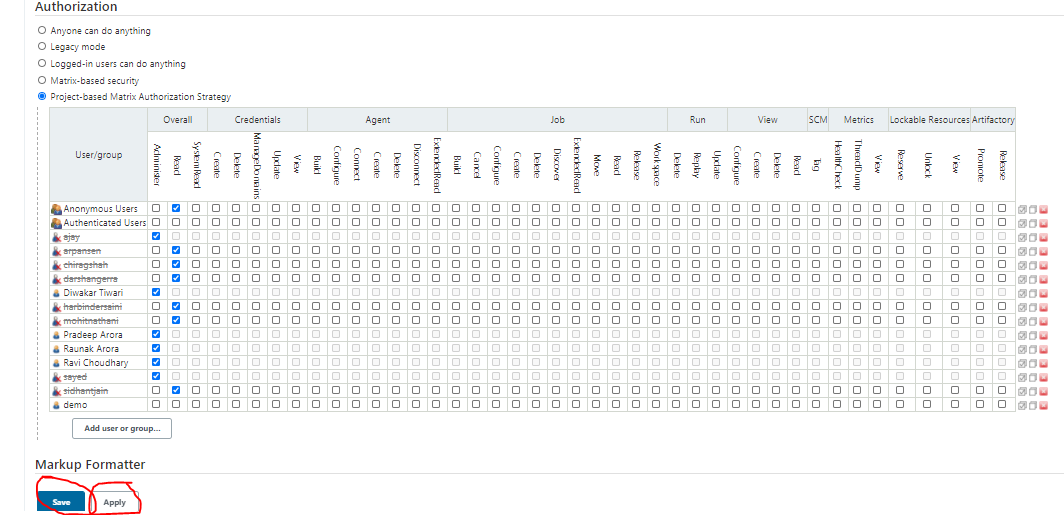
Go to Manage Jenkins > Security >Configure Global Security



Click on Configure Global Security



Select User and Provide Administrator Access.



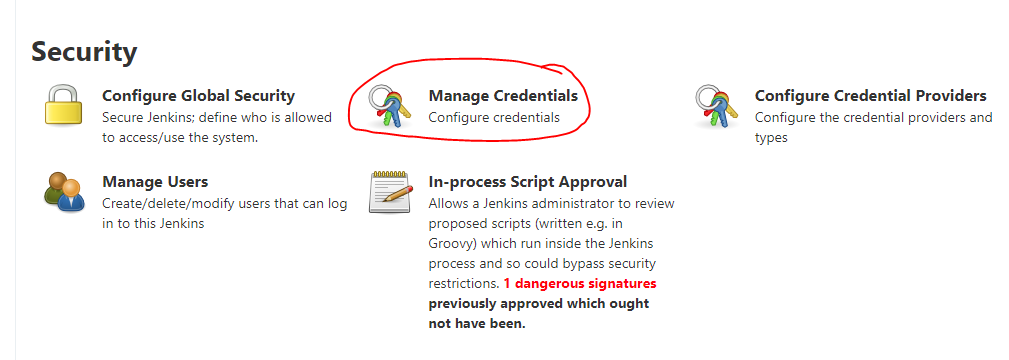
Click on Apply after changes applied save the changes(Click on Save).

After Provide Admin access user can access all jenkins job.

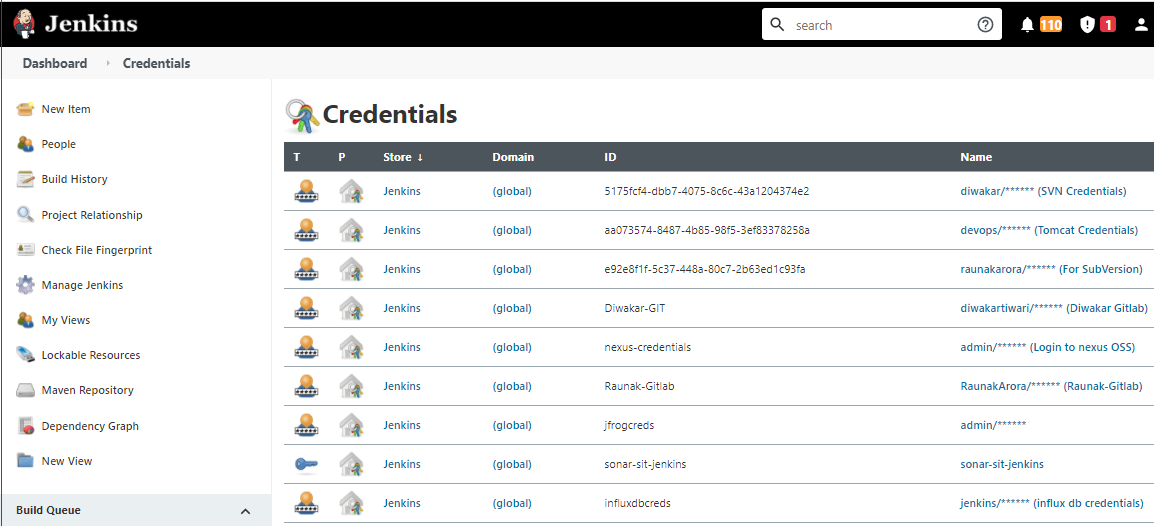
**Credential Creation**

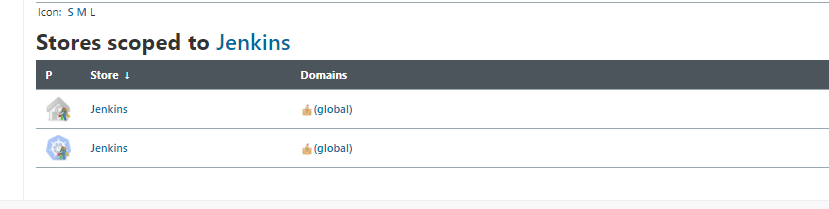
Go to Jenkins Dashboard

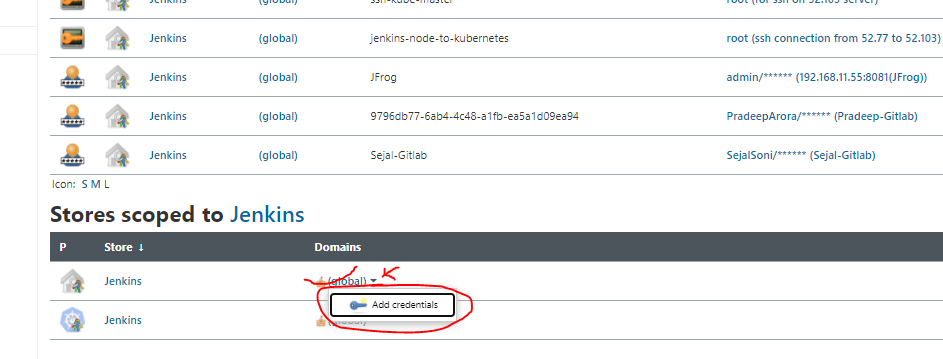
Go to Manage Jenkins > Security >Manage Credentials



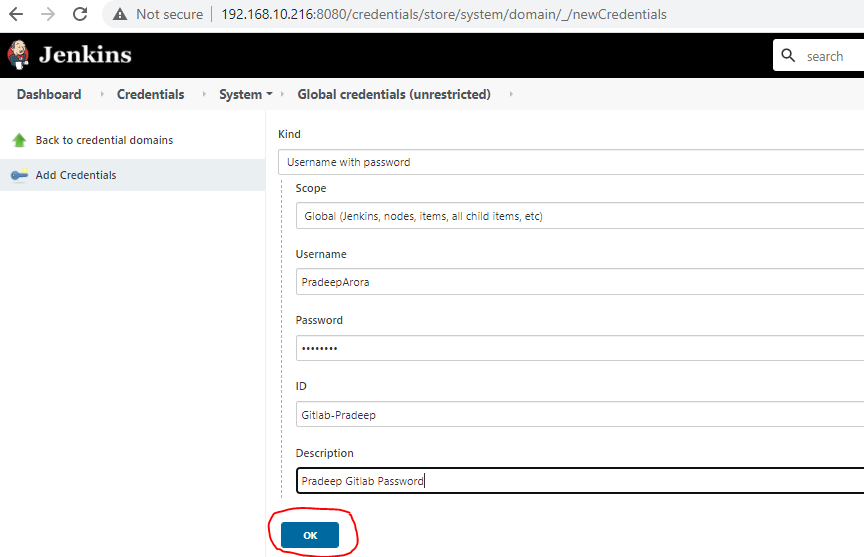
Click on Manage Credentials

****

****

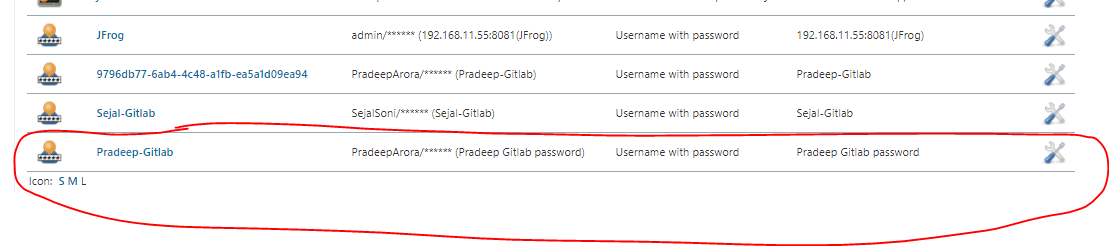
****

At the bottom in Stores Scoped Jenkins -> Click on Arrow of global and click on add Credential



Select Username with password from list and provide username or password like (gitlab, SVN, Jfrog) which credential we want to save provide Unique ID, and Description of password.

Click on OK.

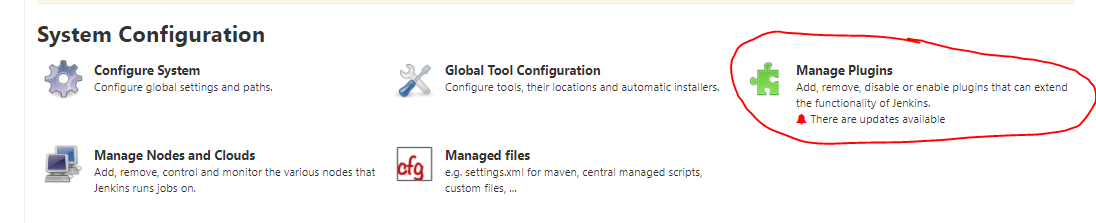


Credential is saved as above snapshot and we can use this credential ID in Jenkins pipeline.

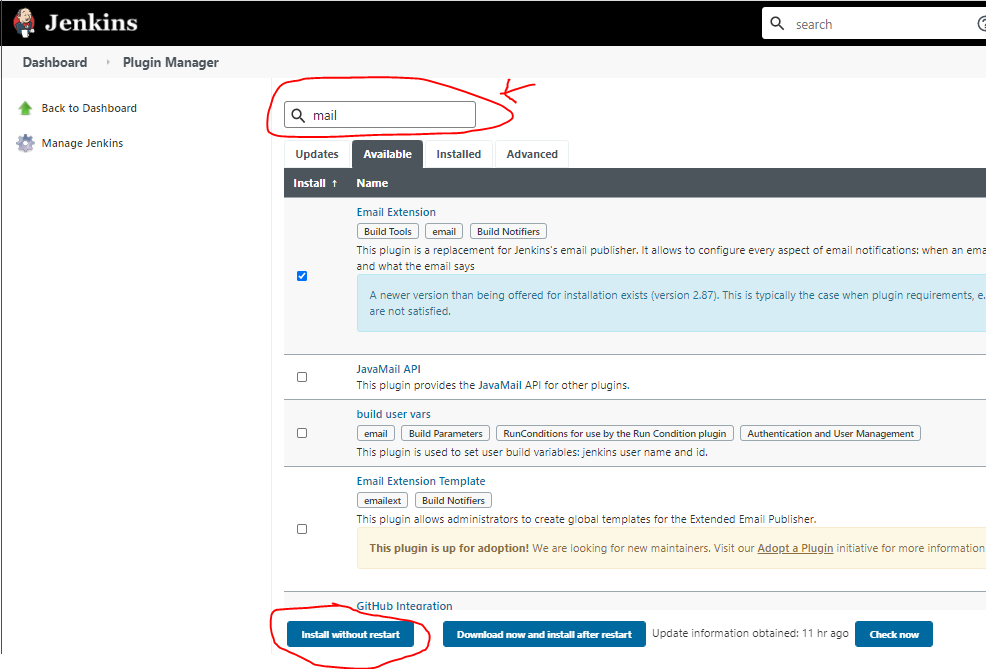
**Manage or Install Plugin**

Go to Jenkins Dashboard

Go to Manage Jenkins > System Configuration >Manage Plugins

****

Click on Manage Plugins



Type Plugin name which we require or need to install.

-> Select Plugin in list

-> click on Install without restart.

-> Plugin will installed and we can use that plugin

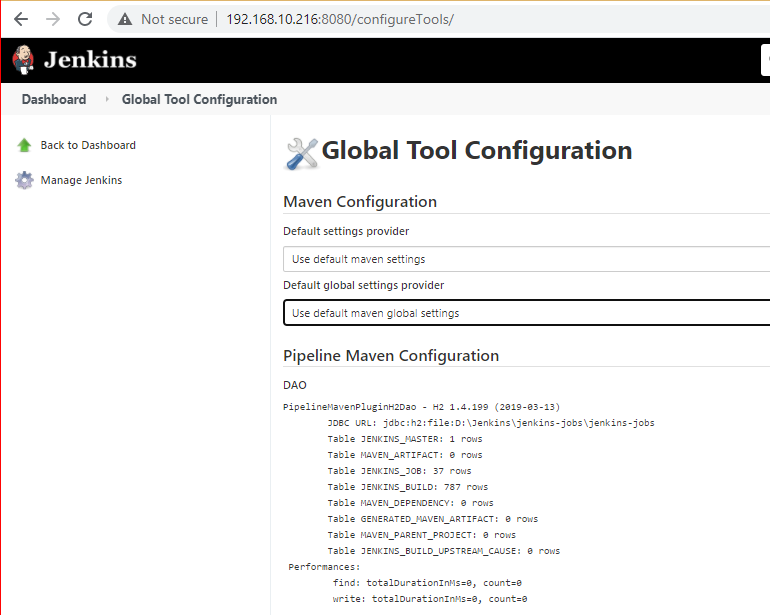
**Tool Configuration in Jenkins**

Here We can Setup tools which we required

Go to Jenkins Dashboard

Go to Manage Jenkins > Global tool configuration >JDK & Maven

**MAVEN Setup**

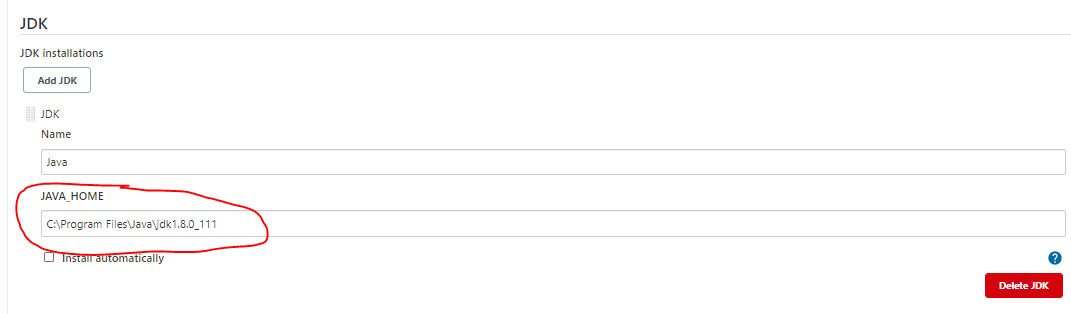


**Git Setup**



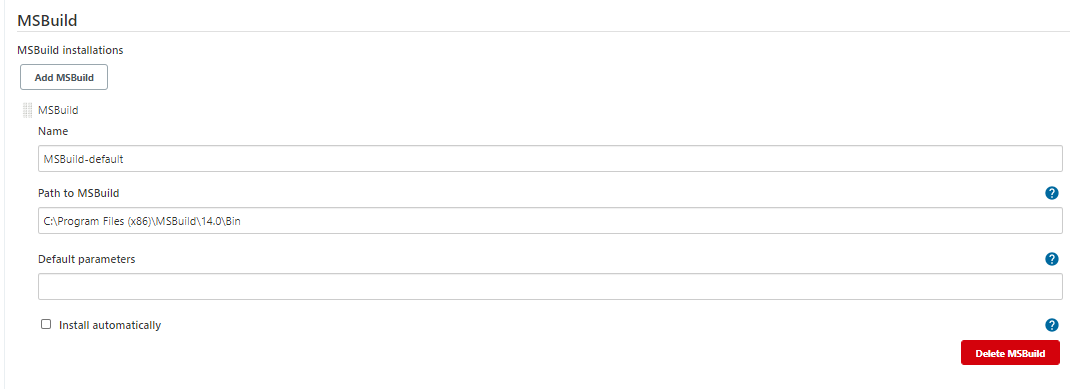
Provide Git path and Save

**Java Setup**



Provide JDK Location and Save.

**MsBuild Setup**



Provide MsBuild Location from jenkins server 10.216 and Save.

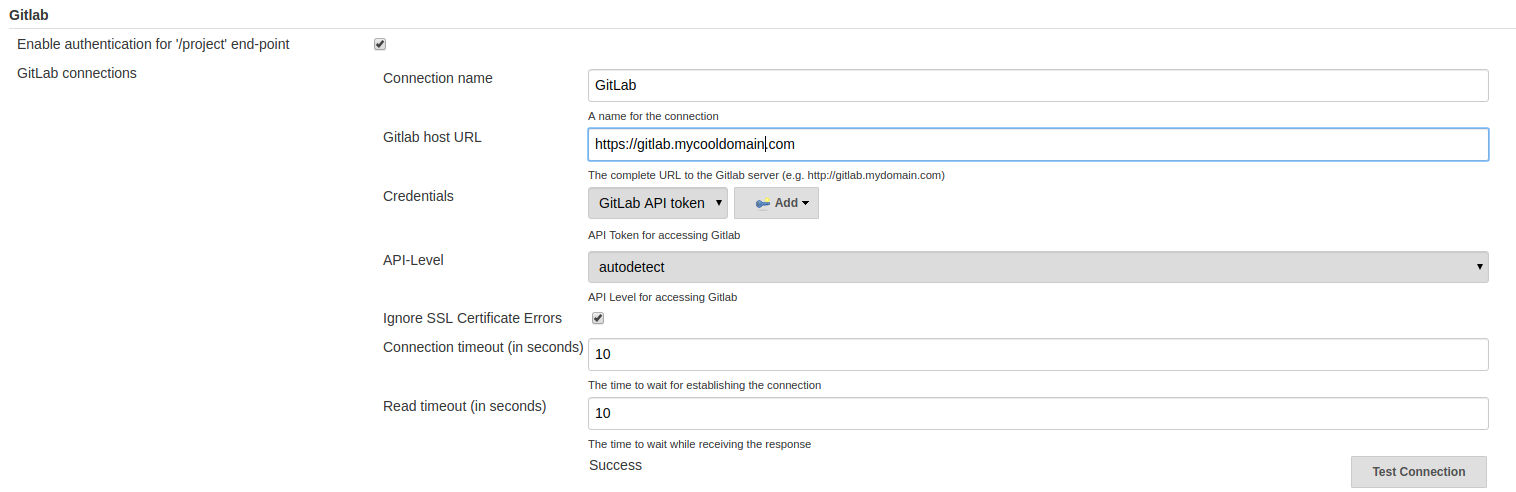
**Jenkins and GitLab integration through webhook**

**Webhook:** A Webhook, also known as a web callback or HTTP push API, is a method for an application to deliver real-time information to other applications. Most APIs involve a request followed by a response. Whereas, no request is required for a webhook; it just delivers data to the applications when it is available. Webhooks become considerably more efficient as a result, for both provider and consumer.

**Requirements for GitLab Webhook Jenkins Integration**

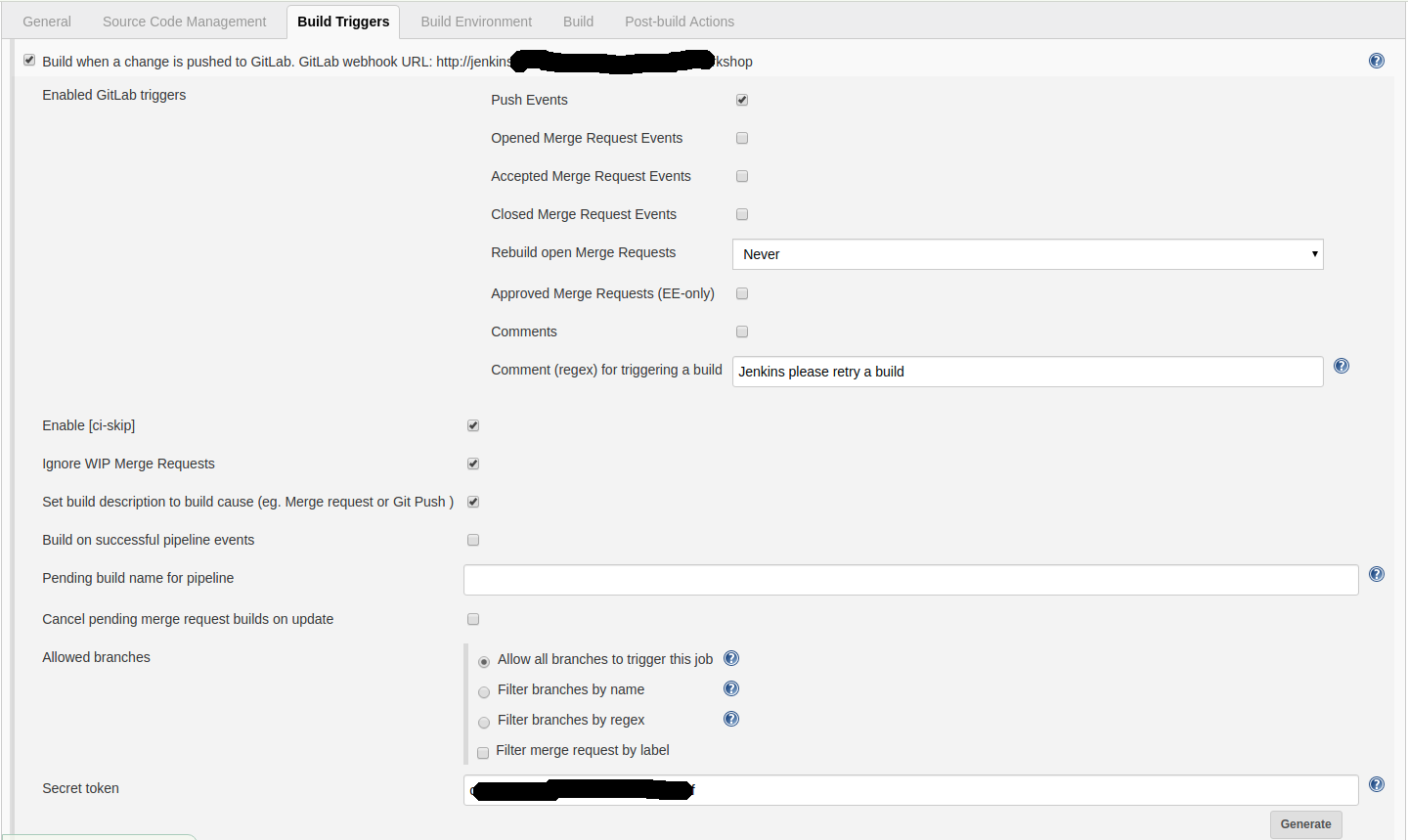
* **Jenkins GitLab Plugin**  
  This plugin allows to build trigger that allows GitLab to trigger Jenkins builds when code is pushed or a merge request is created.
* **Jenkins Git Plugin**  
  This plugin enables use of Git as a build SCM, including repository browsers for numerous providers.
* **Jenkins GitLab Hook plugin**This plugin allows GitLab webhooks to be used to trigger SMC polling on GitLab projects.

Go to **Jenkins>Manage Jenkins>configuration**  
go to the GitLab section and make sure is like

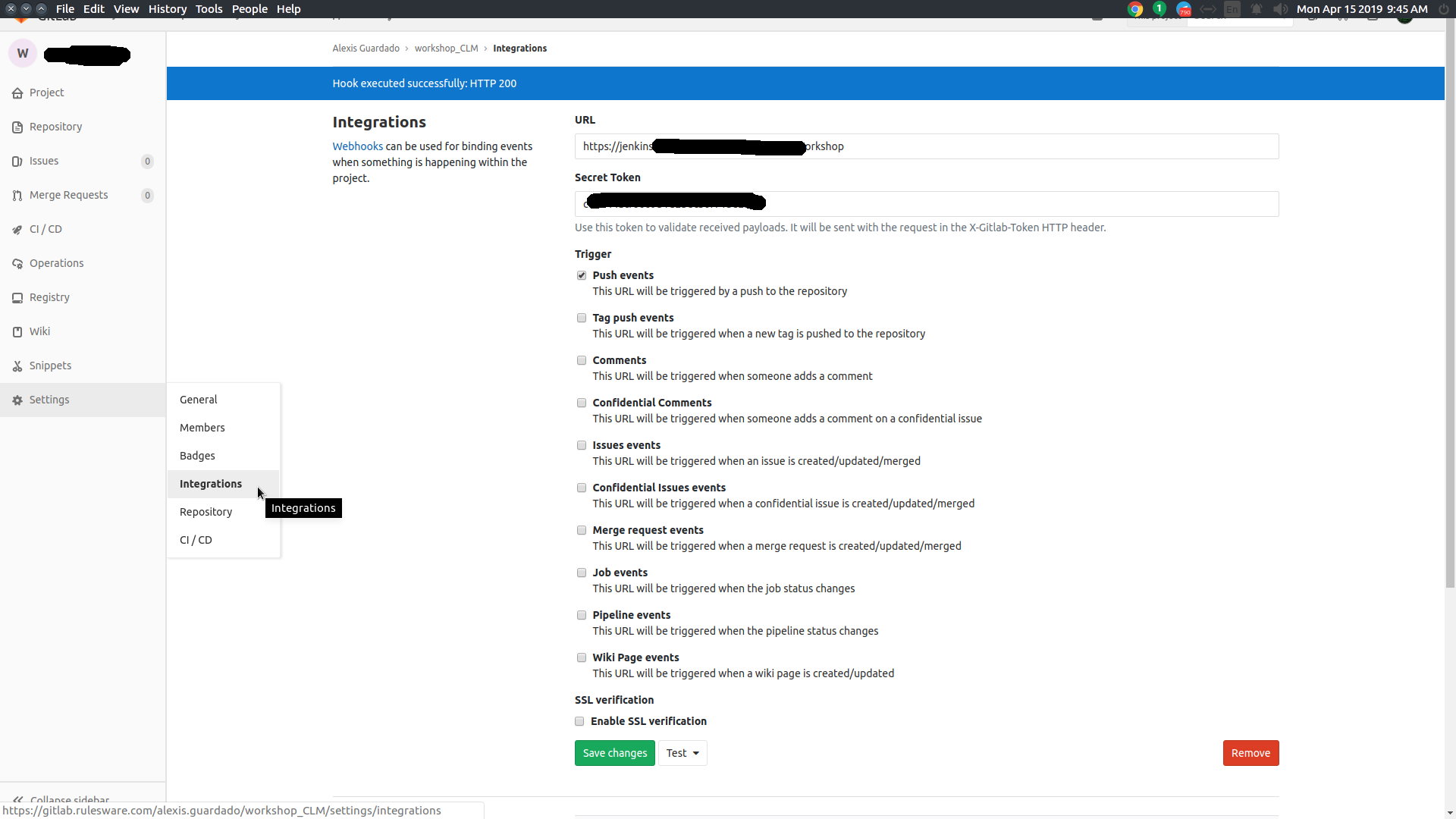


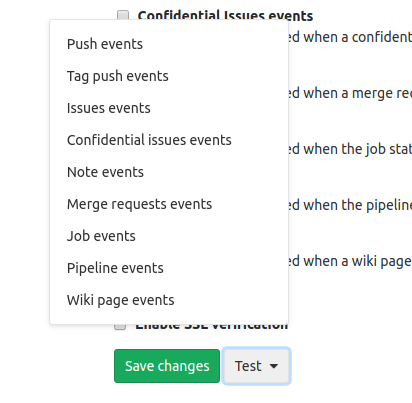
After setting up all this test your connection and make sure is successful.

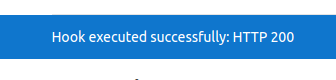
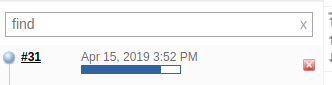
On your Jenkins project go Configure > Build Triggers and select  
Build when a change is pushed to Gitlab. Gitlab webhook URL...  
also, make sure to generate the token in advanced settings



Now in Gitlab go to your Project>Settings>Integrations paste the URL you got in your Jenkins project, as well as the token generated in the Gitlab Build Triggers section, remove ssl Verification

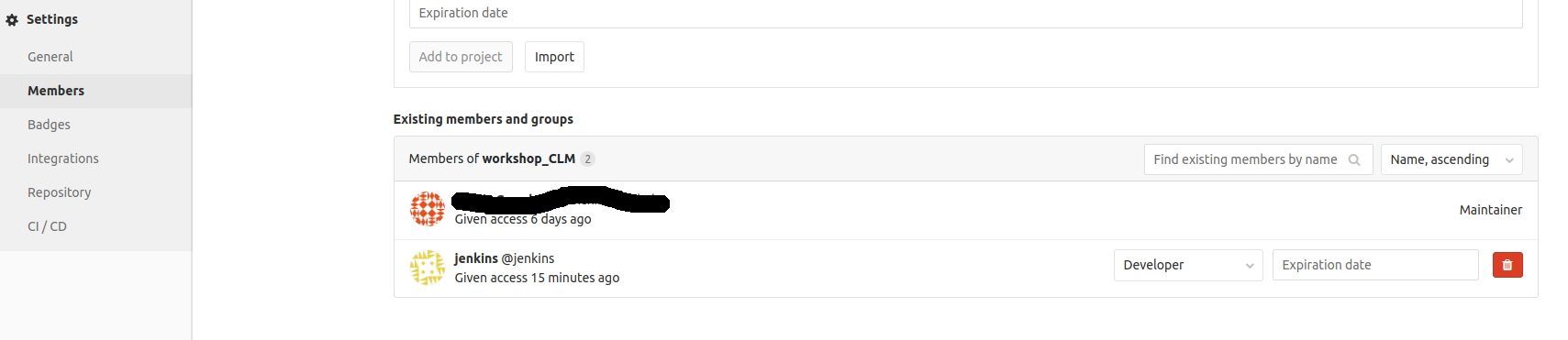


Now hit save and test your hook  


you should get 200 success message on Gitlab and on Jenkins a triggered Job:  
[](https://i.stack.imgur.com/c3S04.png)  


If that's not the case and you still get the 403, this are things you can do:

In Gitlab add a Jenkins user to your project (the one you're trying to get the webhook triggered from). This time it should works.



Troubleshooting

If it still not working another helper you can use is to add a logger to Jenkins go to Jenkins > Manage Jenkins > System log

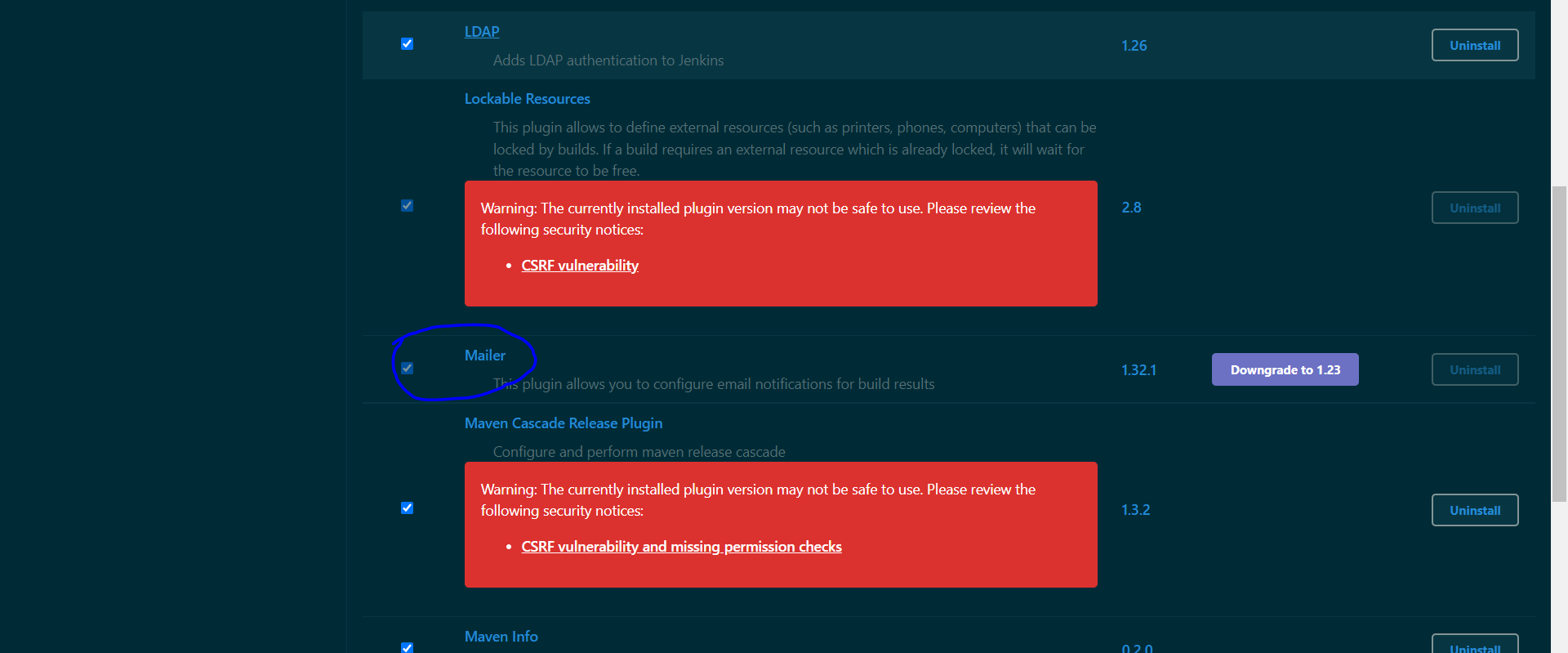
* Add a new log recorder. Type in the name of the logger
* Add the loggers on the input field. Start typing gitlab or webhooks,  
  anything related to Gitlab or webhooks :)

**Mail Server setup in Jenkins**

Click the ‘**Manage Jenkins**’ menu option displayed at the right side of the screen. You will be redirected to the ‘Manage Jenkins’ page, where you need to select the ‘**Manage Plugin**’ option.

Click the ‘**Available**’ tab present at the top of the ‘**Manage Plugin**’ page.

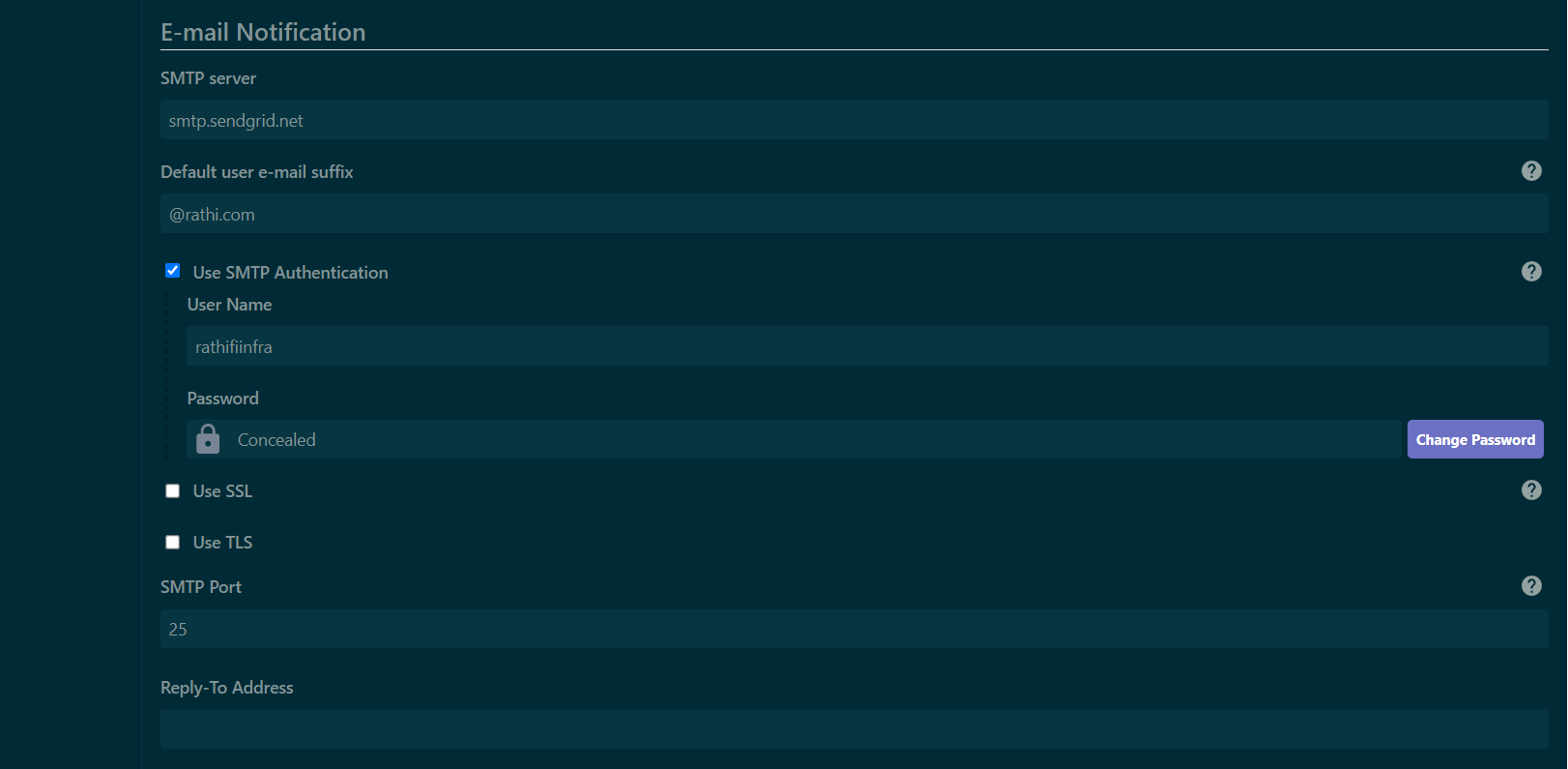
Type ‘**Mailer**’ in the ‘**Filter**’ field displayed at the top-right side of the ‘**Manage Plugin**’ page. Click the checkbox next to the ‘**Mailer**’ option. Click the ‘Install without restart’ button.



Go to the Jenkins home page and click the ‘**Manage Jenkins**’ menu option. Then, select the ‘**Configure System**’ option.

Enter the SMTP server name under ‘**Email Notification**’. Click the ‘**Advanced**’ button and then click the checkbox next to the ‘**Use SMTP Authentication**’ option. Now, set the following fields.

* **SMTP server name** : smtp.sendgrid.net
* **User name**: rathifiinfra
* **Password**: \*\*\*\*\*\*\*\*\*\*\*
* **Use SSL** : Unchecked
* **SMTP Port**: 25



**Cron-Job**

The cron command-line utility, also known as cron job is a [job scheduler](https://en.wikipedia.org/wiki/Job_scheduler) on [Unix-like](https://en.wikipedia.org/wiki/Unix-like) [operating systems](https://en.wikipedia.org/wiki/Operating_system). Users who set up and maintain software environments use cron to schedule jobs (commands or [shell scripts](https://en.wikipedia.org/wiki/Shell_script)) to run periodically at fixed times, dates, or intervals.

**Basic Syntax: \* \* \* \* \***

**First \* for minute (0-59)**

**Second \* for hour (0-12)**

**Third \* for day of month (1-31)**

**Forth \* for month (1-12)**

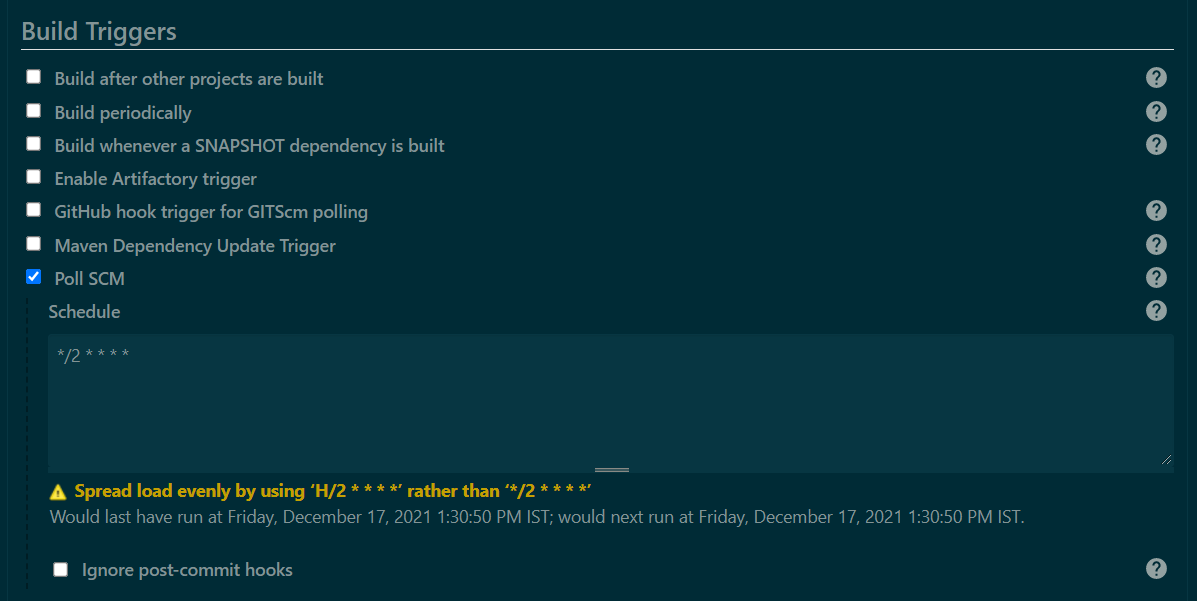
**Fifth \* day of week (0-6) Sunday to Saturday 7 is also mention as Sunday.**

**Schedule a sample job**

In the job configuration page, let's scroll down straight to the ***Build Triggers*** section. Since we intend to create a straightforward job, let's select the checkbox marked ***Build periodically***. As soon as we select this checkbox, a Text Box is displayed with the ***Schedule*** label.

**We have to provide value in a**[Cron-compliant format](https://www.baeldung.com/cron-expressions). There's extensive information available on the page if we click the question mark next to the box.

Let's type ***\*/2 \* \* \* \**** here, which represents an interval of two minutes:



We can see information right beneath the box. It tells us about when the job will run next.

**Pipeline Defined**

Pipelines are Jenkins jobs enabled by the Pipeline (formerly called “workflow”) plugin and built with simple text scripts that use a Pipeline DSL (domain-specific language) based on the Groovy programming language.

Pipelines leverage the power of multiple steps to execute both simple and complex tasks according to parameters that you establish. Once created, pipelines can build code and orchestrate the work required to drive applications from commit to delivery.

**Declarative Pipeline Vocabulary**

In declarative pipeline terms such as “Agent,” “stages,” “stage” and “steps” are a subset of the vocabulary used for Jenkins in general.

**Declarative pipeline is started with pipeline tag and below is the steps we used in pipeline:**

**Agent:** In declarative pipelines the **Agent** directive is used for specifying which agent/slave the job/task is to be executed on. This directive only allows you to specify where the task is to be executed, which agent, slave, and label or Docker image.

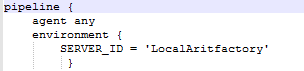
**Stages:**Containing a sequence of one or more [stage](https://www.jenkins.io/doc/book/pipeline/syntax/#stage) directives

**Stage:**the stages section is where the bulk of the "work" described by a Pipeline will be located. At a minimum, it is recommended that stages contain at least one [stage](https://www.jenkins.io/doc/book/pipeline/syntax/#stage) directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy.

**Steps:**The steps section defines a series of one or more [steps](https://www.jenkins.io/doc/book/pipeline/syntax/#declarative-steps) to be executed in a given stage directive.

**Process of CICD JAVA Project:**

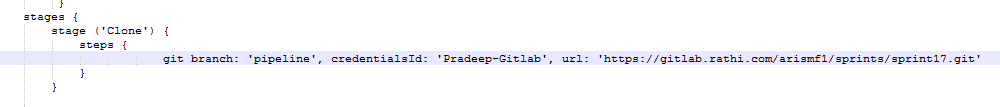
**Jenkins\_File we use for process:**

****

Define Jenkins Anent

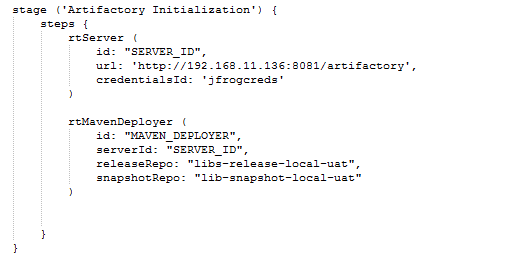
**Step 1:**

Stage - Clone

**→ In this step we initialize the git branch and pull the code from gitlab.**

**Step 2:**

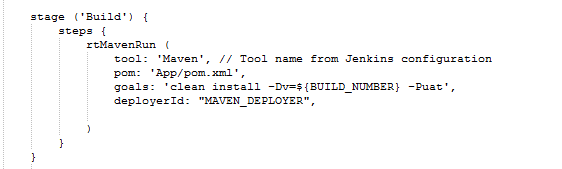
stage ('Artifactory Initialization')

****

**→ In this step we initialize the Artifactory, Meven\_Deployer and Maven\_resolver.**

**Step 3:**

Stage - Build



**→ In this step we create build of code with build number.**

**Step 4:**

stage ('JUnit Test') {

steps {

tool: 'Maven', // Tool name from Jenkins configuration

pom: 'pom.xml',

goals: 'jacoco:prepare-agent install',

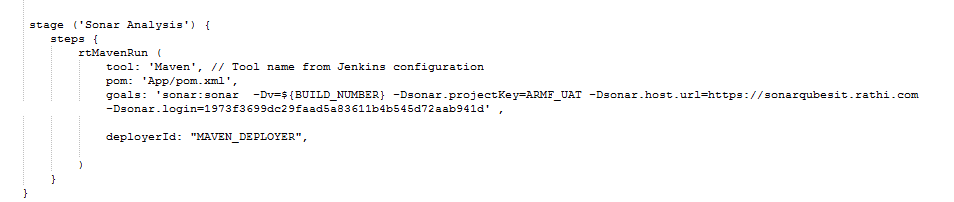
}

}

**→ In this step we perform JUnit test on code.**

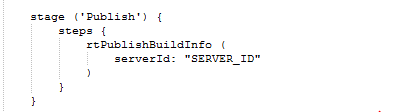
**Step 5:**

Stage : Sonar Anylysis

**→ In this space we generate SonarQube report the code for code quality.**

**Step 6:**

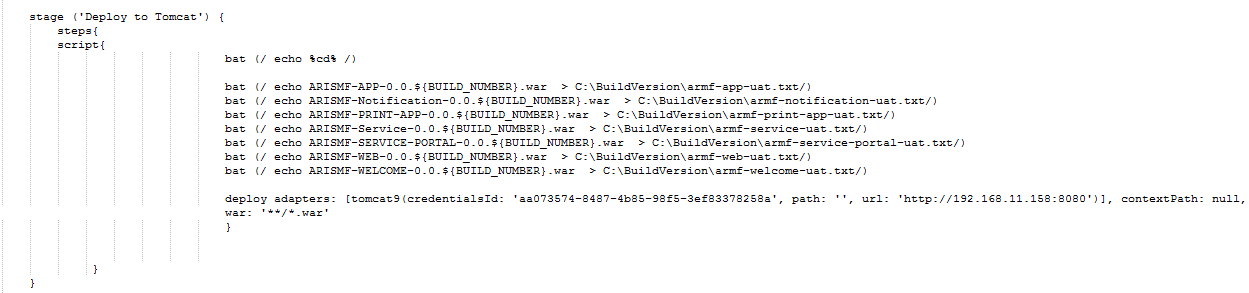
Stage : Publish



**→ In this step we publish the code on artifactory server with build number**

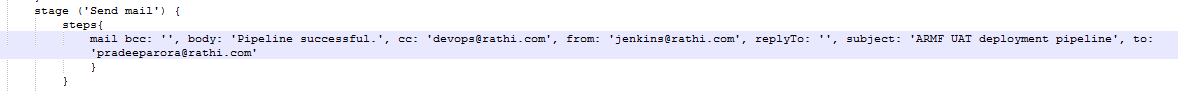
**step 7:**

Stage : Deploy to Tomcat



**→ In this step we deploy the code on Tocmat server and save the .war file name in txt file for code revert script with build number.**

**Step 8:**

**→ In this step we sent an alert mail to user for build success.**

**Dependencies for JAVA CICD pipeline:**

**First:** We need to set git on Jenkins server for pulling the code from GitLab. Also we need to install git plugin. And set global credential of GIT user for using in CICD pipeline.

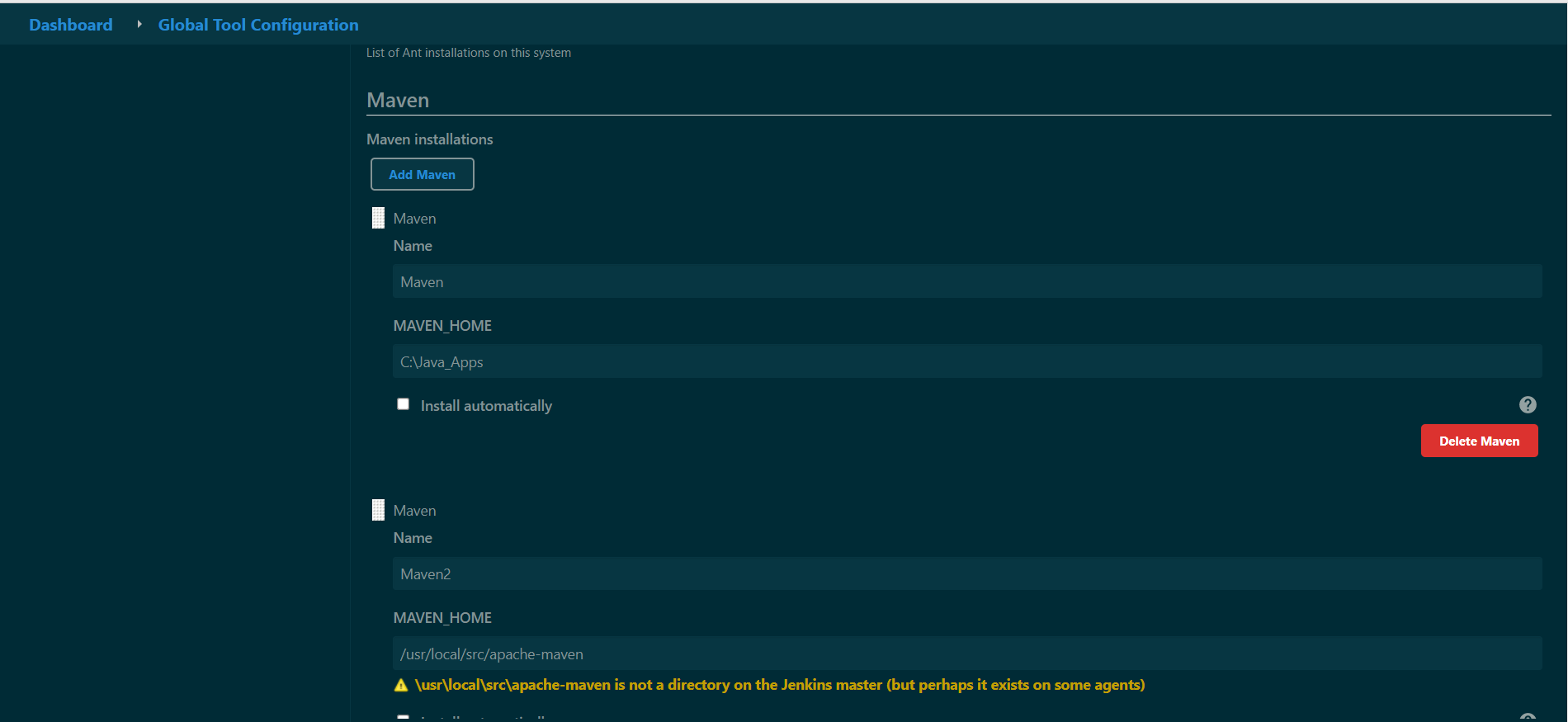
**Plugin: Git**.

**Second:** For 'Artifactory Initialization’ we need to setup JFrog Artifactory and setup global user for access Artifactory also install plugin for the artifactory.

**Plugin: Artifactory**.

**Third:** For building the JAVA code we need to setup Maven on Jenkins server and install maven plugin in Jenkins. Also we need to setup Global tool configuration for Maven tool.

**Plugin: Maven Integration**



**Forth:** For JUnit step for java project we need to install JUnit plugin in Jenkins. Also we need to configure Jococo plugin in POM.xml file for the particular project.

**Plugin: JUnit**



**Fifth:** For setup SonarQube in CICD we need to setup Global Tool Configuration for SonarQube Scanner. Also setup SonarQube plugin and dependency in POM.xml file for particular project.

**Plugin: SonarQube Scanner**

<dependency>

<groupId>org.sonarsource.scanner.maven</groupId>

<artifactId>sonar-maven-plugin</artifactId>

<version>3.7.0.1746</version>

</dependency>

<plugins>

<plugin>

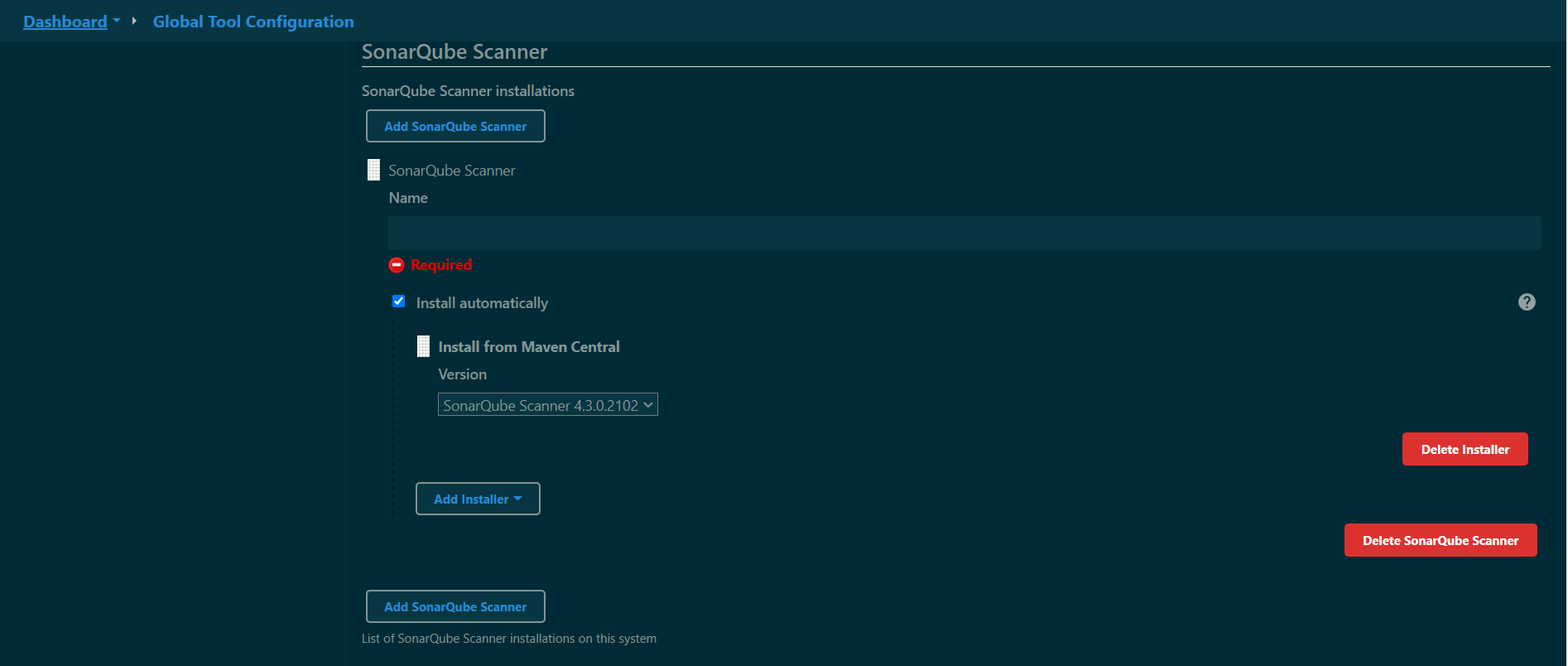
<groupId>org.sonarsource.scanner.maven</groupId>

<artifactId>sonar-maven-plugin</artifactId>

<version>3.4.0.905</version>

</plugin>

</plugins>



**Sixth:** For publish build code on Artifactory we need to setup global credential for the JFrog Artifactory access.

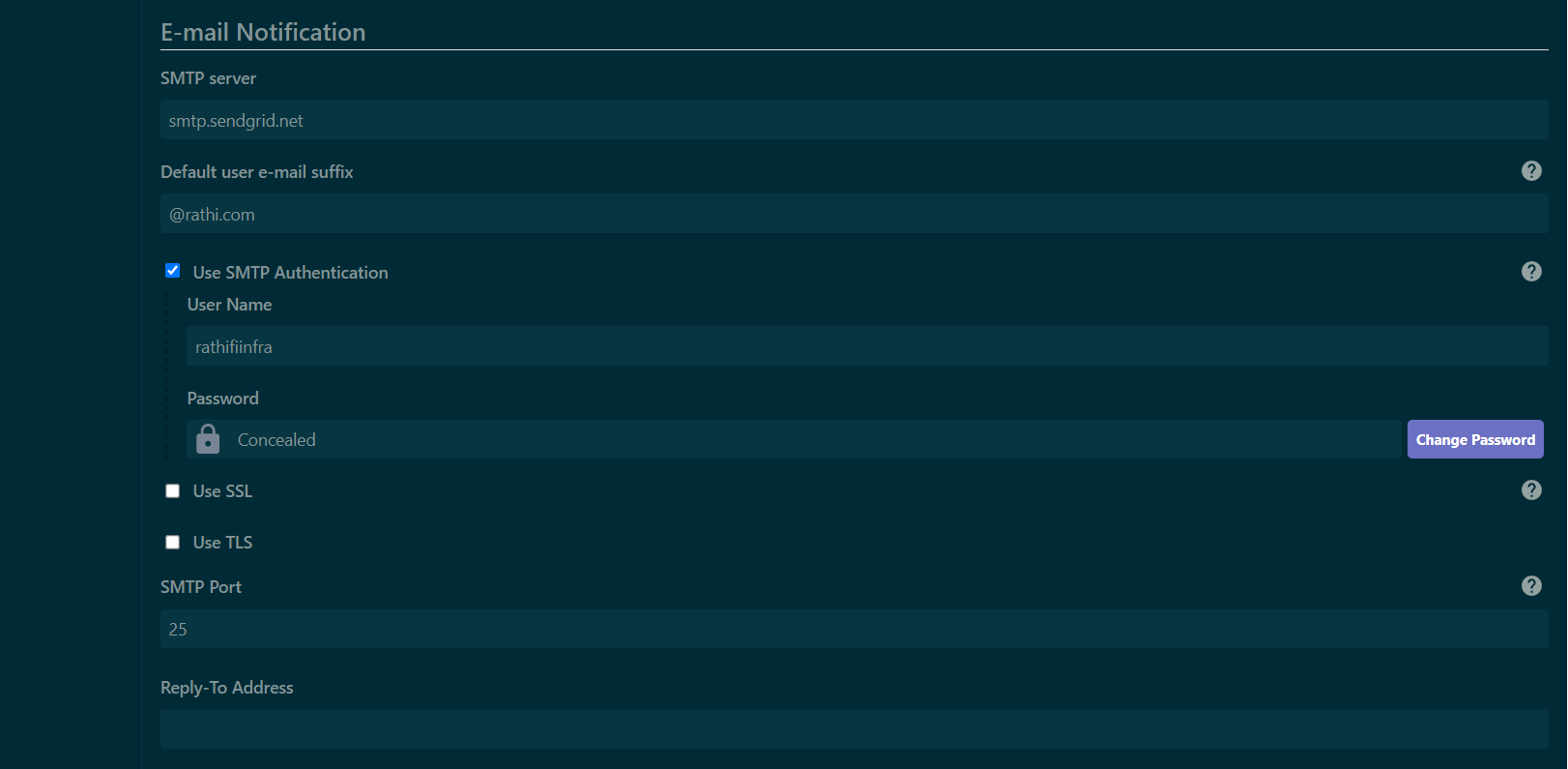
**Plugin: Artifactory**.

**Seventh**: For deploying code on Tomcat, We need tomcat installed on application where we need to deploy code, also have a tomcat-manger user with admin access for deployment through Jenkins. And we need global credential setup for tomcat manager for particular application server.

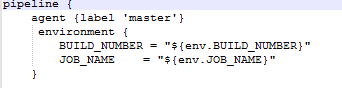
**Plugin:** [**Deploy to container Plugin**](https://plugins.jenkins.io/deploy)

**Eighth:** For sending the mail confirmation to respected team we need to setup mail server for the same.

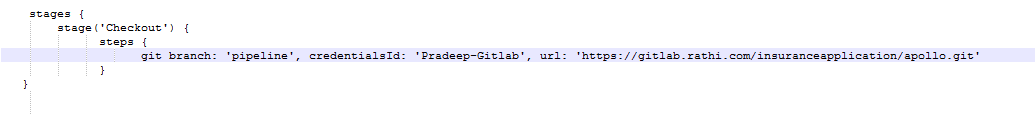
**Plugin:** [**Mailer**](https://plugins.jenkins.io/mailer)



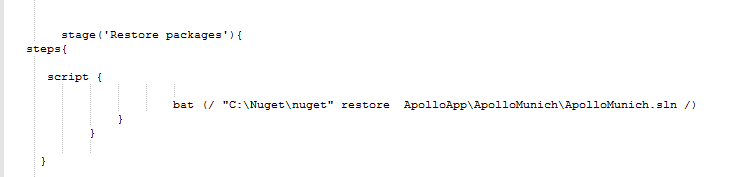
**.NET CICD Pipeline**

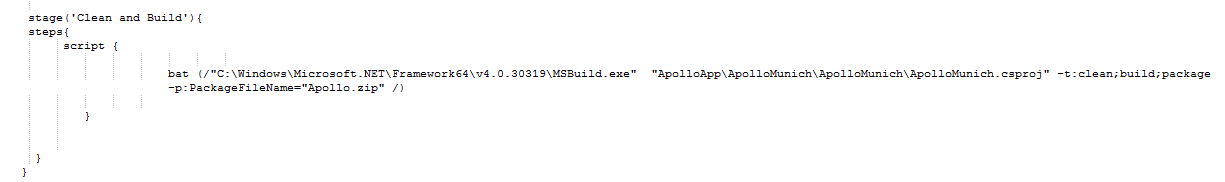


**First: In this step we start declarative pipeline and set build\_number and job\_name variable for storing on local artifactory.**



**Second: In this step we initialize the git branch and pull the code from gitlab.**

 **Third: In this step we install the required nuget packages for running our application. For installation of nuget packages and build the project we need to setup msbuild on jenkins server and setup globel tool configuration for msbuild tool and also need to install MSBuild plugin.**

**Forth: In this step we build the project using msbuild and create zip file for the build project for deploying through ms-deploy tool.**

stage ('Sonar Analysis') {

steps {

script {

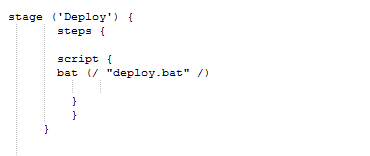
bat (/ "sonartest.bat" /)

}

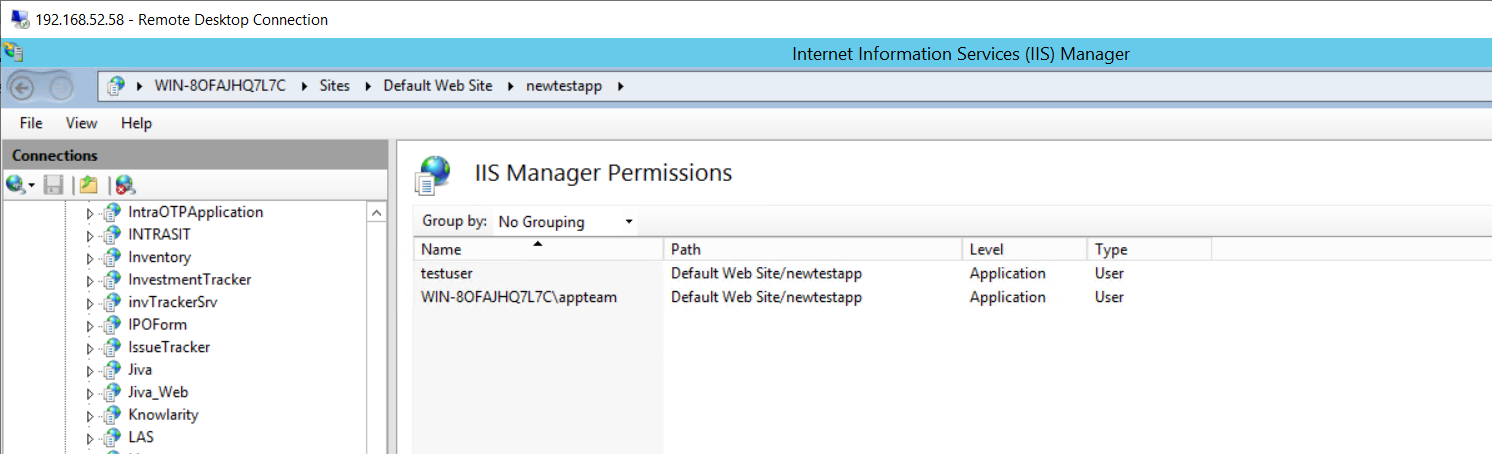
}

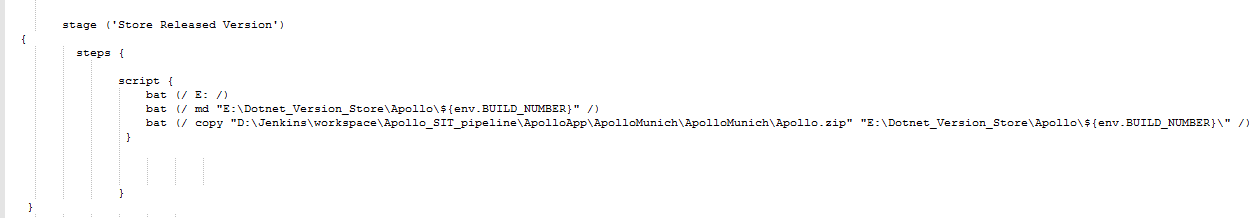
}

**Fifth: In this step we need to setup SonarQube scanner on Jenkins server and need msbuild for the scan.**



**Sixth: For this step we need to install msdeploy.exe on application server and provide access to user with full access in IIS Manager Permission on the required project.**





**Seventh: In this step we store artifactory in local system in form of .ZIP file on Drive E.**

stage ('Test Case') {

steps {

script{

bat (/ D: /)

bat (/ cd "D:\Test-Case\Project-name" /)

}

git branch: 'master', credentialsId: 'Raunak-Gitlab', url: 'https://gitlab.rathi.com/arit-automation/gmc\_employee\_sit.git'

rtMavenRun (

tool: 'Maven', // Tool name from Jenkins configuration

pom: 'pom.xml',

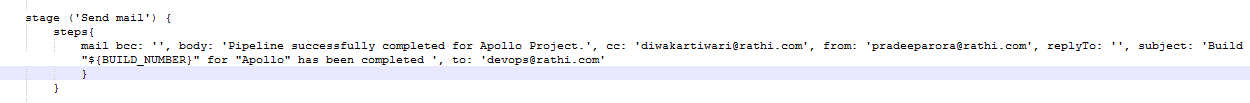
goals: 'clean install',

)

}

}

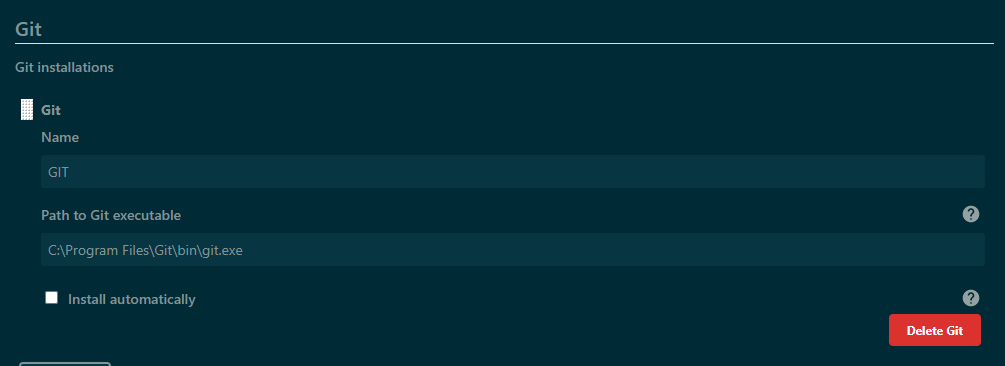
**Eight: In this step we run selenium test cases by using create build of selenium code with Maven, Selenium test cases written by Testing team.**

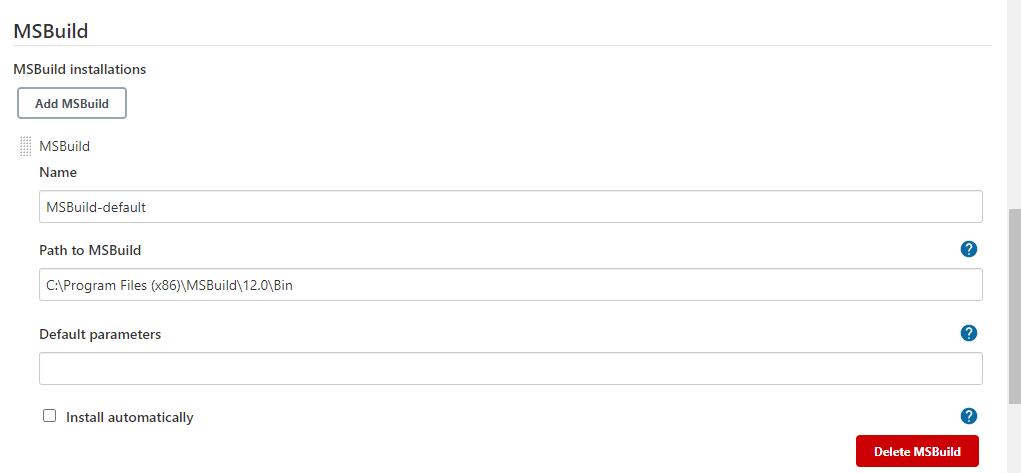
**Ninth: In this step we sent confirmation mail to respected team or PM using Mailer plugin.**

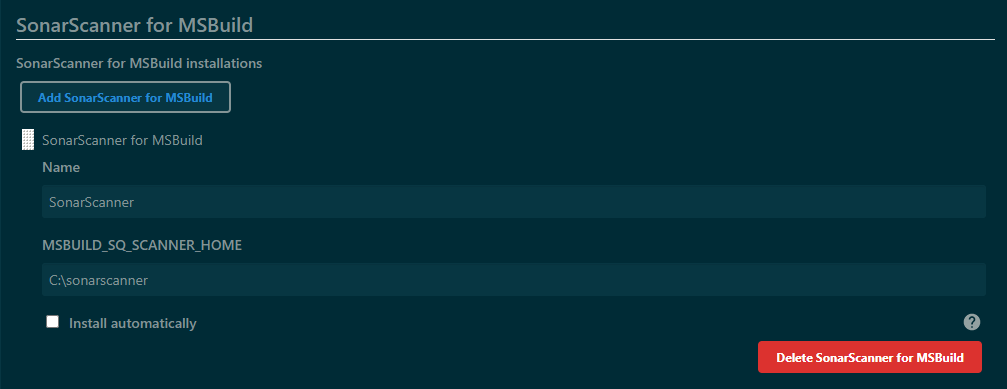
**Jenkins .Net Project Requirment**

* Install below pulgin from Dashboard> Manage Jenkins > Manage Plugins
* Gitlab
* MsBuild
* Pipeline
* [Pipeline: Declarative Agent API](https://plugins.jenkins.io/pipeline-model-declarative-agent)
* Build Pipeline
* SonarQube Scanner (and put sonar scanner on your jenkins server)
* Set Path For Gitlab and MsBuild and sonar Scanner

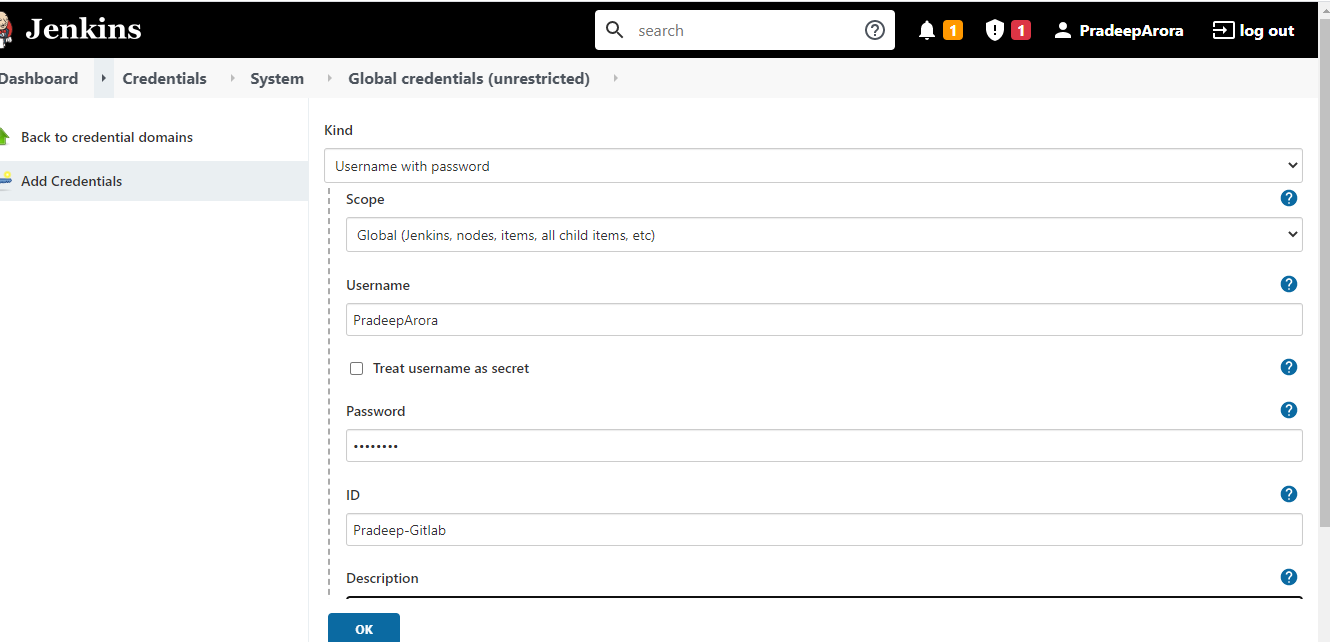
Intall Gitbash and Visual Stdio on Jenkins server and provide path Dashboard > Gitlab Tools Configuration

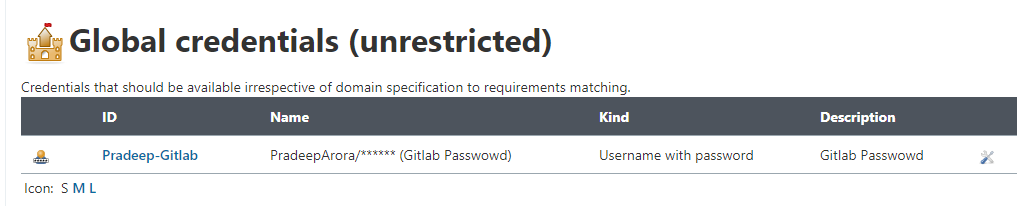




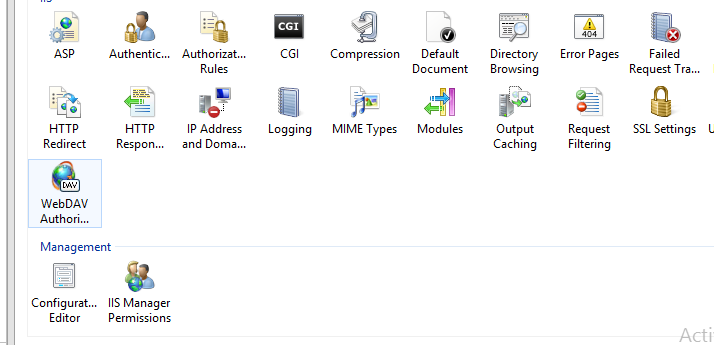


* Setup Gitlab login credential in Credential manager tool.

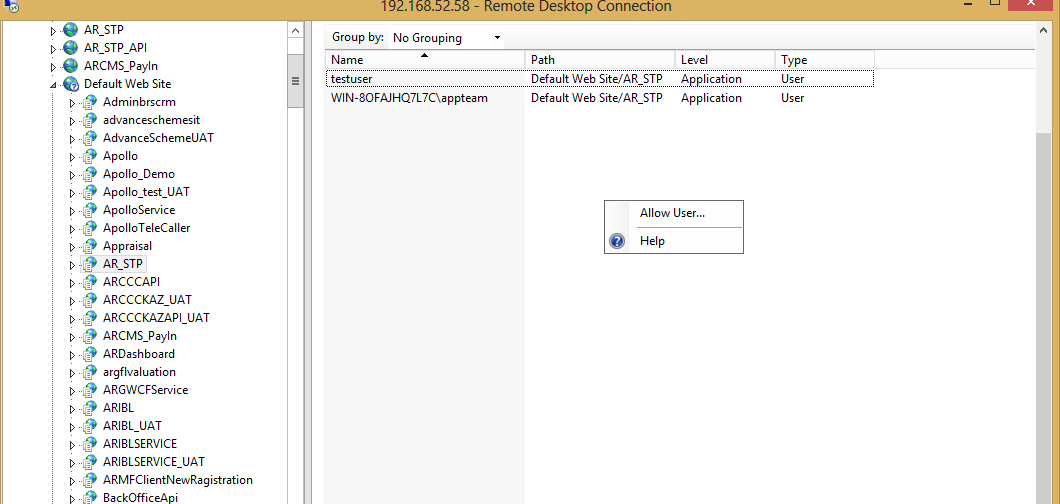




* Using this credential-ID in pipeline
* Create a Pipeline .
* Create Deploy.bat and Sonarscanner.bat file for deploy and sonarqube anylysis.
* Pipeline Document given in other file.
* Create User on IIS server for access of application



* Click on IIS Manager Permission
* Create User



Allow user and create a new one and give this credentials in Deploy.bat file

**.net project CICD structure**

* Create project on sonarqube
* Create a project on IIS with empty folder
* Install msdeploy and web platform on server where we want to deploy application
* Install Ms Deploy Latest version
* Install Web Platform latest version
* Check web Ms Deployer
* **C:\Program Files\IIS\Microsoft Web Deploy**
* Create IIS Manager User.
* **Managment -> IIS Manager User -> Add User -> Give User Name and Password.**
* Give Permission to user on application.
* **IIS Manager Permission.**
* Provide full permission to User and Application.
* **IIS User**
* **IUSER**
* **Interctive**
* Go To Jenkins create a new CICD pipeline and copy old running pipeline and configure according project.