**Table of Contents**

[1. INTRODUCTION 5](#_Toc124781818)

[**2. INSTALLATION OF JENKINS** 6](#_Toc124781819)

[2.1 Installation on Windows 6](#_Toc124781820)

[2.1.1 Pre-Requisites 6](#_Toc124781821)

[2.1.2 Installation Steps 6](#_Toc124781822)

[2.1.3 Post Installation Steps 7](#_Toc124781823)

[2.2 Installation on Linux 9](#_Toc124781824)

[2.2.1 Pre-Requisites 9](#_Toc124781825)

[2.2.2 Installation Steps 10](#_Toc124781826)

[2.2.3 Post Installation Steps 11](#_Toc124781827)

[3. FIRST JOB IN JENKINS 12](#_Toc124781828)

[**4. CONFIGURATION GUIDELINES FOR JENKINS** 17](#_Toc124781829)

[4.1 Following are the guidelines for developers: 17](#_Toc124781830)

[4.1.1 Backup Jenkins home regularly 17](#_Toc124781831)

[4.1.2 Use fingerprinting to manage dependencies. 17](#_Toc124781832)

[4.1.3 The most reliable builds will be clean builds, which are built fully from source code control. 17](#_Toc124781833)

[4.1.4 Always configure job to generate trend reports and automated testing while running builds. 17](#_Toc124781834)

[4.1.5 Set up Jenkins on the partition that has the maximum free disk-space 18](#_Toc124781835)

[4.1.6 Setup different job for each maintenance or development branch we create. 18](#_Toc124781836)

[4.1.7 Allocate different ports for parallel project builds and avoid scheduling all jobs to start at same time. 18](#_Toc124781837)

[4.2 Following are the guidelines for Jenkins Admin/Lead: 18](#_Toc124781838)

[4.2.1 Monitoring Jenkins. 18](#_Toc124781839)

[4.2.2 Securing Jenkins. 19](#_Toc124781840)

[4.2.3 Viewing & keeping logs. 19](#_Toc124781841)

[4.2.4 Authenticating scripted clients 19](#_Toc124781842)

[4.2.5 Integrate Jenkins with issue tracking system like JIRA to reduce the need for maintaining 19](#_Toc124781843)

[change log 19](#_Toc124781844)

[4.2.6 Archive unused jobs before removing them. 20](#_Toc124781845)

[4.2.7 Setup email notifications mapping to all developer in the project, so that everyone on the team has his pulse on project’s current status. 20](#_Toc124781846)

[**4.3 Following are the best practices for Jenkins Pipeline:** 20](#_Toc124781847)

[**4.3.1 Use the Jenkins Community Contributed Pipeline Plugin** 20](#_Toc124781848)

[**4.3.2 Develop Your Pipeline as Code** 20](#_Toc124781849)

[4.3.3 Do all work within a stage 21](#_Toc124781850)

[4.3.4 All tasks & stages should be done in Nodes only 22](#_Toc124781851)

[4.3.5 Use withEnv instead of env global variable for setting variables 22](#_Toc124781852)

[4.3.6 Prefer stashing files to archive them 23](#_Toc124781853)

[5. INTEGRATION 24](#_Toc124781854)

[5.1 Integration with SonarQube 24](#_Toc124781855)

[5.2 Integration with Email 28](#_Toc124781856)

[5.3 Integration with Slack 33](#_Toc124781857)

[6. NAMING CONVENTION TO BE FOLLOWED IN JENKINS 37](#_Toc124781858)

[7. BUILD CONFIGURATION FOR MASTER/SLAVE ARCHITECTURE 38](#_Toc124781859)

[8. REFERENCE LINKS 40](#_Toc124781860)

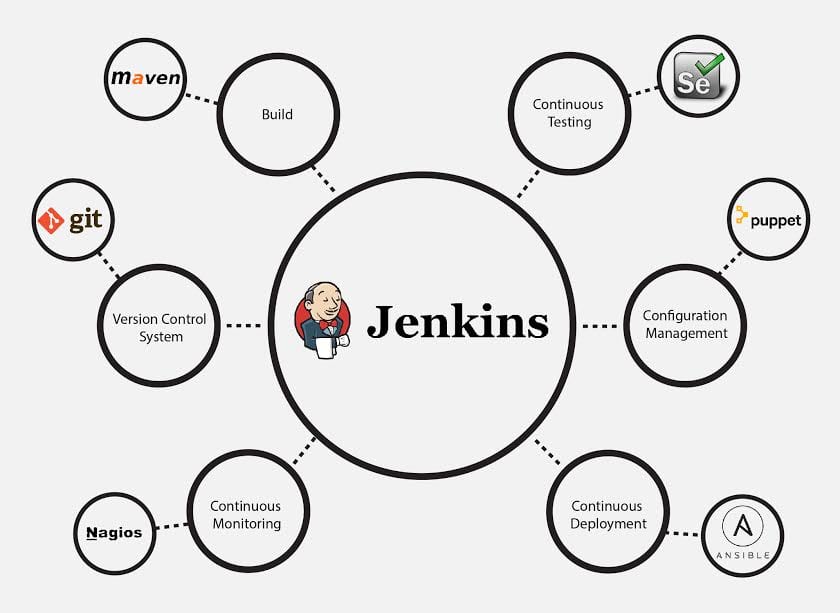
# INTRODUCTION

Jenkins is an open-source automation tool written in Java with plugins built for continuous integration. 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.

The image below depicts that Jenkins is integrating various DevOps stages:



1. **INSTALLATION OF JENKINS**

## Installation on Windows

### Pre-Requisites

* Minimum Hardware requirements:

1. 256 MB of RAM
2. 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

* System Requirements:

1. 8 GB + of RAM
2. 50 GB+ of drive space

* Software requirements:

1. Java: see the [Java Requirements](https://www.jenkins.io/doc/administration/requirements/java) page | Java 8 or Java 11
2. Web browser: see the [Web Browser Compatibility](https://www.jenkins.io/doc/administration/requirements/web-browsers) page
3. For Windows operating system: [Windows Support Policy](https://www.jenkins.io/doc/administration/requirements/windows)

* Ref Link: <https://www.jenkins.io/doc/book/installing/war-file/>

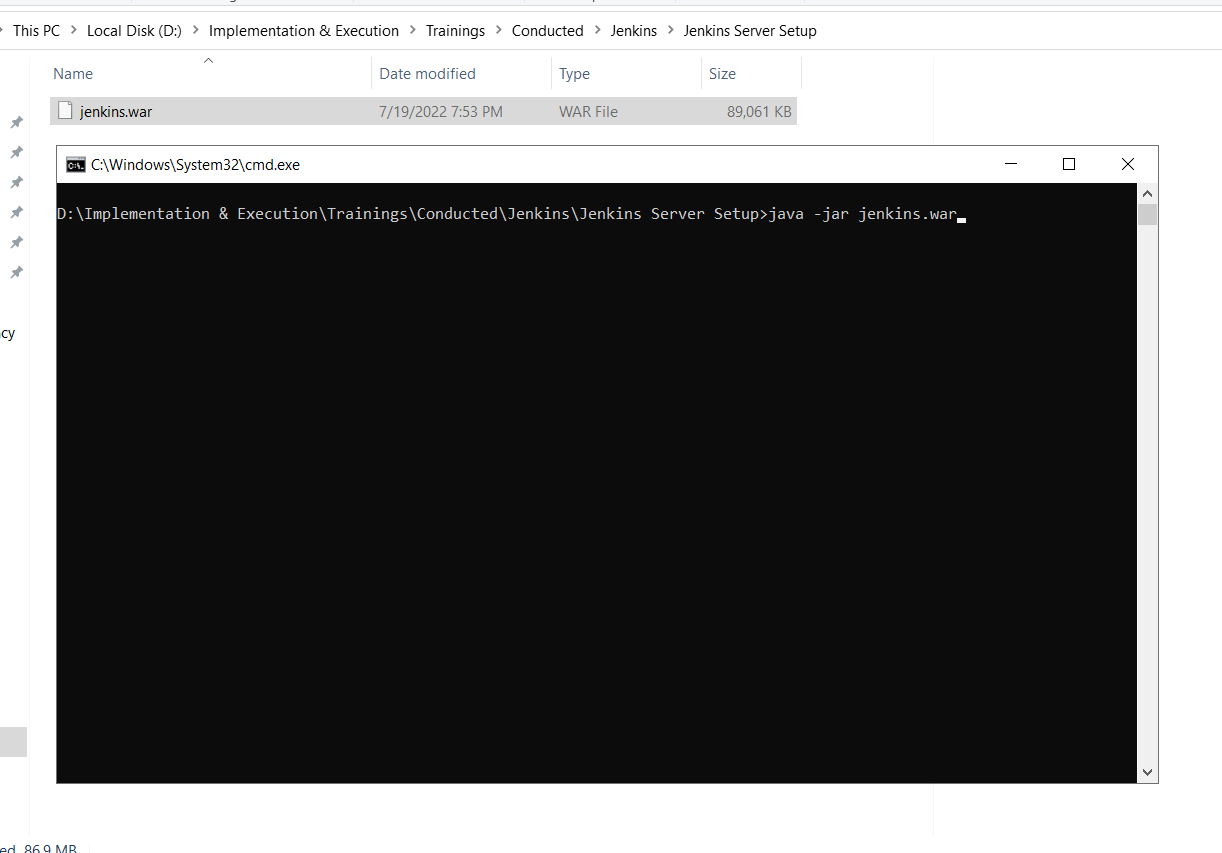
### Installation Steps

1. Download the Jenkins.war file from below link.

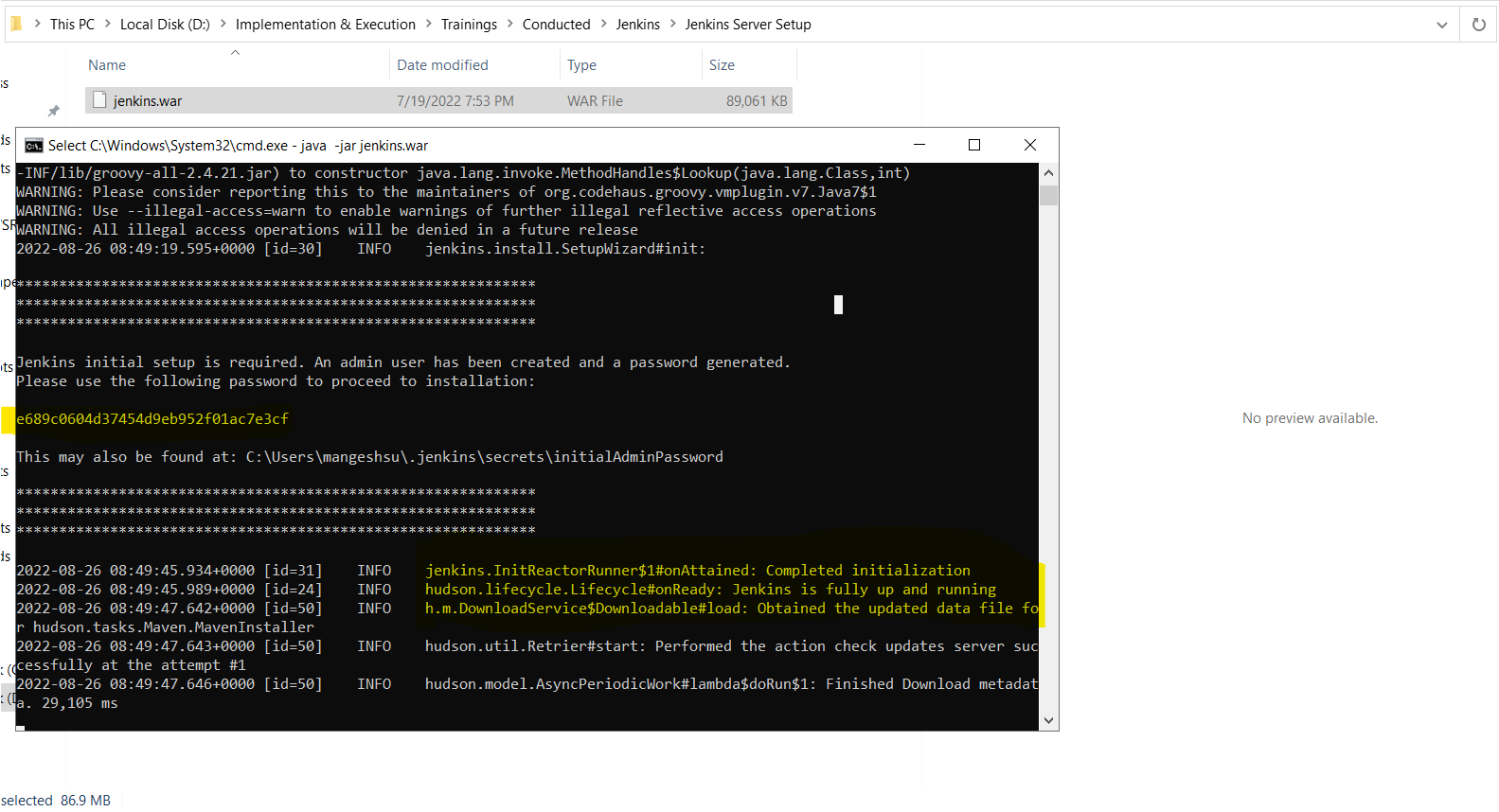
<https://www.jenkins.io/download/#downloading-jenkins>

1. Run below command using CMD in the directory containing Jenkins.war:

Java -jar Jenkins.war



1. Copy Initial Admin Password once server is up from command prompt.

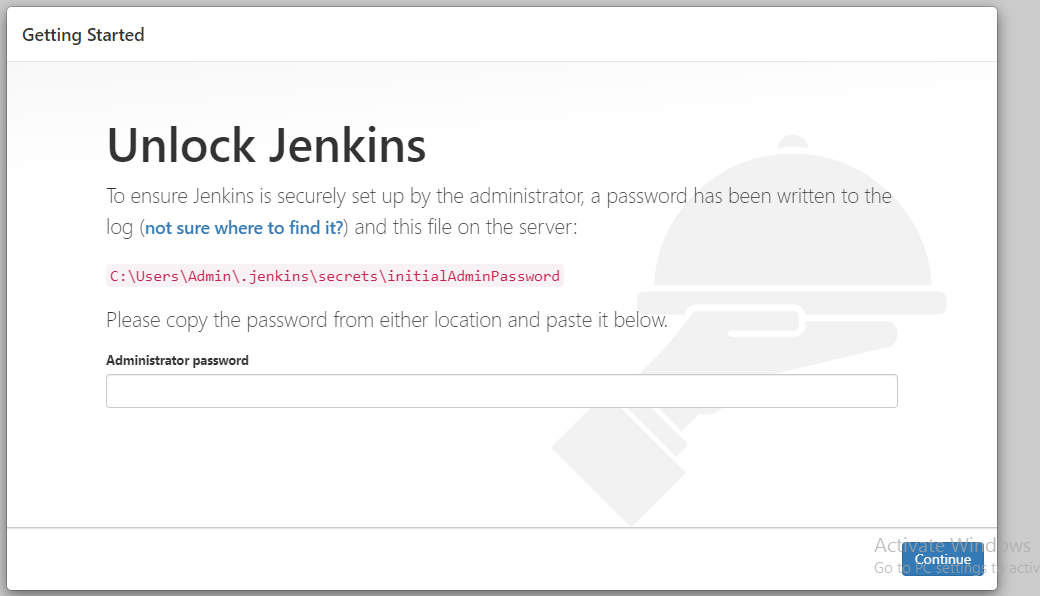


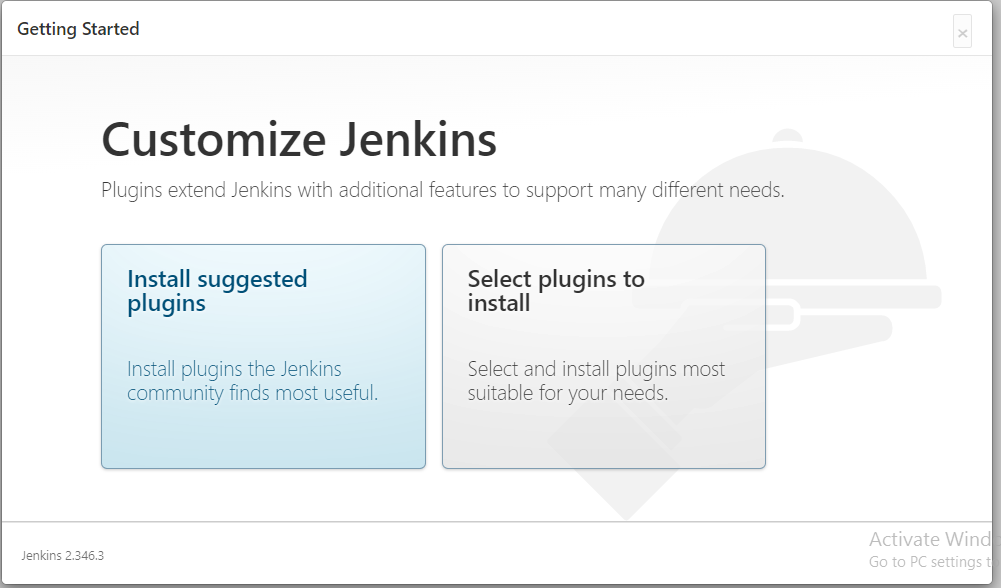
**Note:**

* + 1. In-Order to start Jenkins server please re-run command mentioned in 2nd step.
    2. Please don’t close Command Prompt once Jenkins server started.
    3. Please Complete Post installation setup once Jenkins server up & running.

### Post Installation Steps

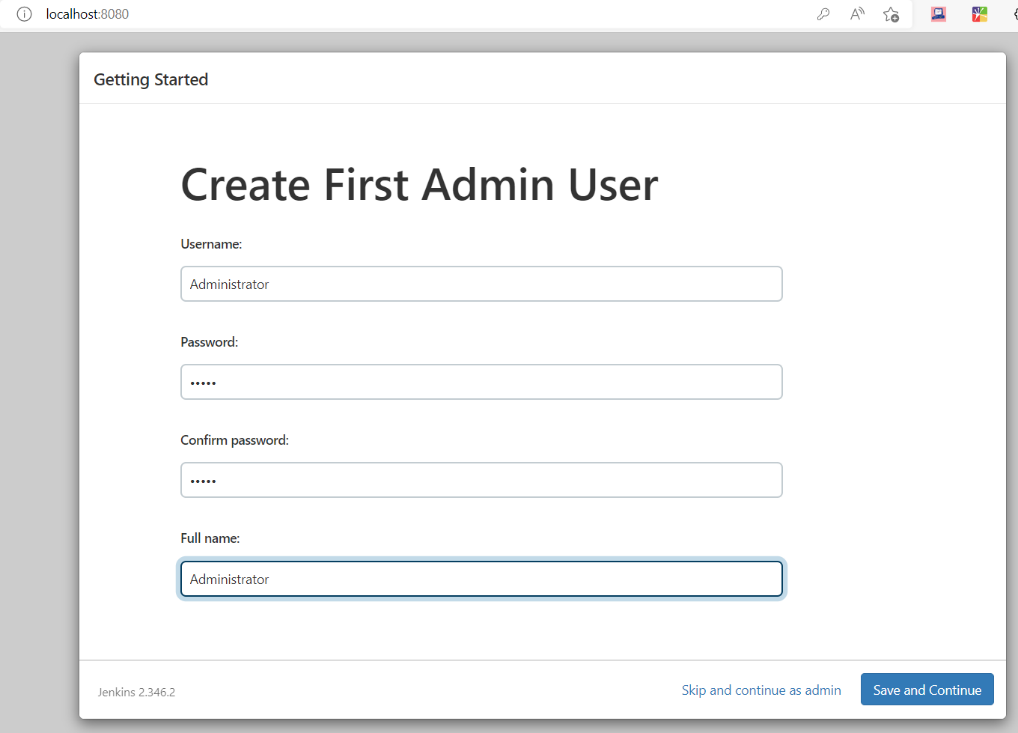
Open <http://localhost:8080/> in browser & follow next steps.

1. ****Enter previously copied InitialAdminPassword or copy it from location mentioned in below Fig.

1. ****Continue with Installation of suggested plugin.

**Note:**

* 1. In Order to Install suggested plugins, the update center URL’s should be Accessible.
  2. Installation of suggested plugins is must.
  3. Continue with first admin user account creation.

****

## Installation on Linux

### Pre-Requisites

* Minimum Hardware requirements:

1. 256 MB of RAM

2. 1GB of drive space (although 10 GB is a recommended minimum if running Jenkins as

Docker container)

* System Requirements:

1. 8 GB + of RAM

2. 250 GB+ of drive space

* Software requirements:

1.Java: see the [Java Requirements](https://www.jenkins.io/doc/administration/requirements/java) page | Java 8 or Java 11

2.Web browser: see the [Web Browser Compatibility](https://www.jenkins.io/doc/administration/requirements/web-browsers) page

3.For Windows operating system: [Windows Support Policy](https://www.jenkins.io/doc/administration/requirements/windows)

* Ref Link: <https://www.jenkins.io/doc/book/installing/war-file/>

### Installation Steps

The Installation Steps for Jenkins on a Linux Operating System.

**Note:** Perform all the commands as a **root** user or use **Sudo.**

1. Installing wget package

$ yum install wget -y

2. Add Jenkins repository to Linux

$ wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

3. Import GPG repository key

$ rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

$ amazon-linux-extras install epel -y

4. Update the list of repositories to confirm it is working

$ yum update -y

$ sudo yum repolist

5.Install java 11 version for latest version of jenkins

$ sudo yum install java-11-openjdk -y

6.Check java version

$ java --version

7. Install jenkins

$ yum install jenkins -y

8. Start and enable the jenkins service to start at the OS boot

$ systemctl status jenkins

$ systemctl start jenkins

9. Access Jenkins Server on Amazon Linux 2

http://[serverip\_ip\_or\_hostname]:8080

10.The default login password is store in this file:

$ sudo cat /var/lib/jenkins/secrets/InitialAdminPassword

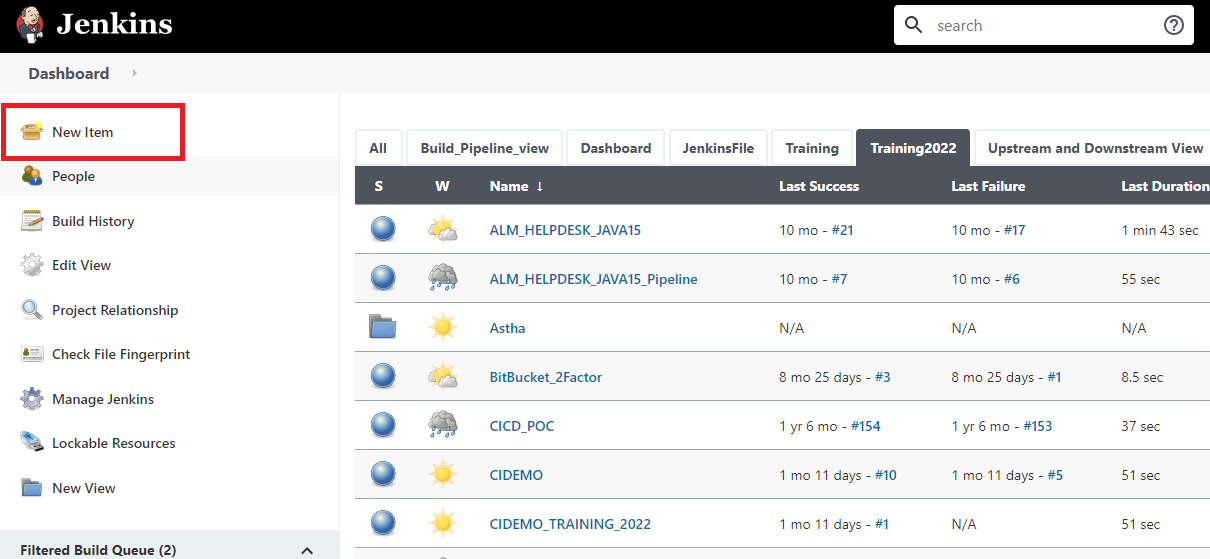
### Post Installation Steps

Post Installation, the steps are same as mentioned in Windows Installation.

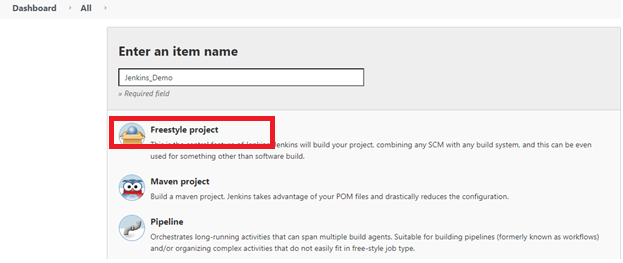
# FIRST JOB IN JENKINS

Once installation has been done, go to Jenkins dashboard.

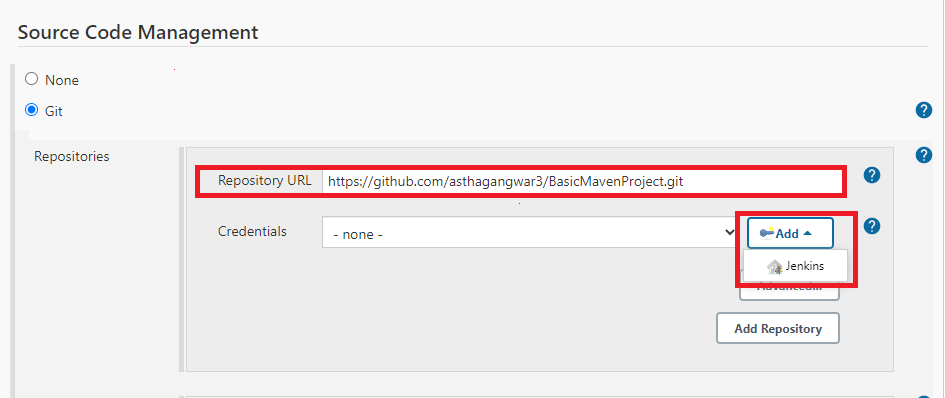
Click on **New item**.



Choose **Freestyle Project** and give it a meaningful name to it and Click **Ok**.



In the **Source Code Management section,** provide Repository URL.

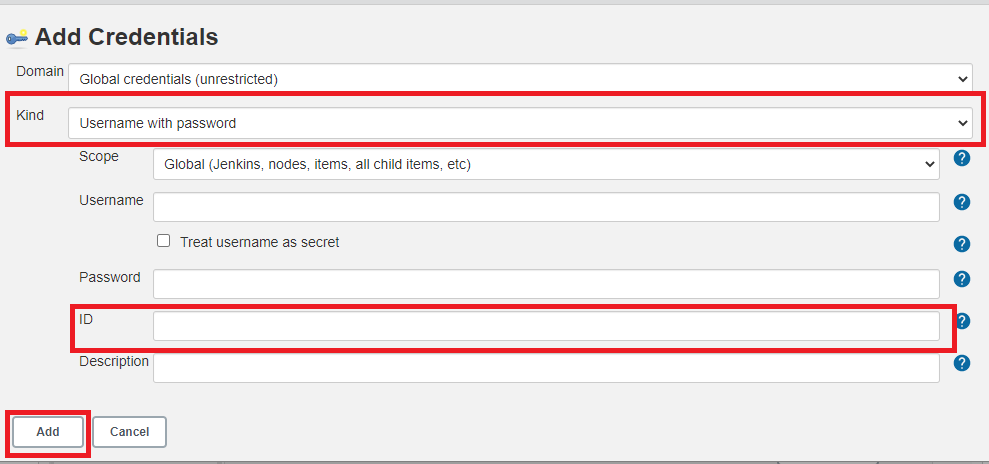
****

If it is **a private repository**, you have to provide the credentials.

In order to add the credentials, click on **Add button** and then **Jenkins** option.

* Choose **Kind 🡪 Username and password.**
* Provide Username and Password of your Source Code Management Tool. (e.g. GitHub).
* Give meaningful ID.
* Then, click on **Add button** to store credentials.

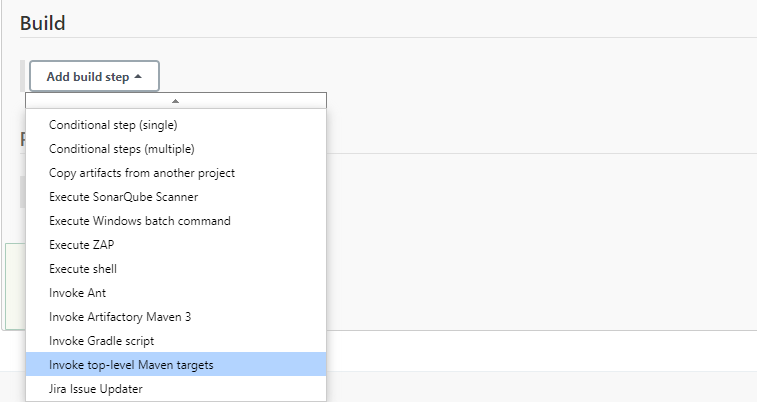
**Note:** There are various Kind Options available, choose it according to your requirement.



Now go to **Build** Section**.**

Application is a maven-based application, hence choosing **“Invoke top-level Maven target”**

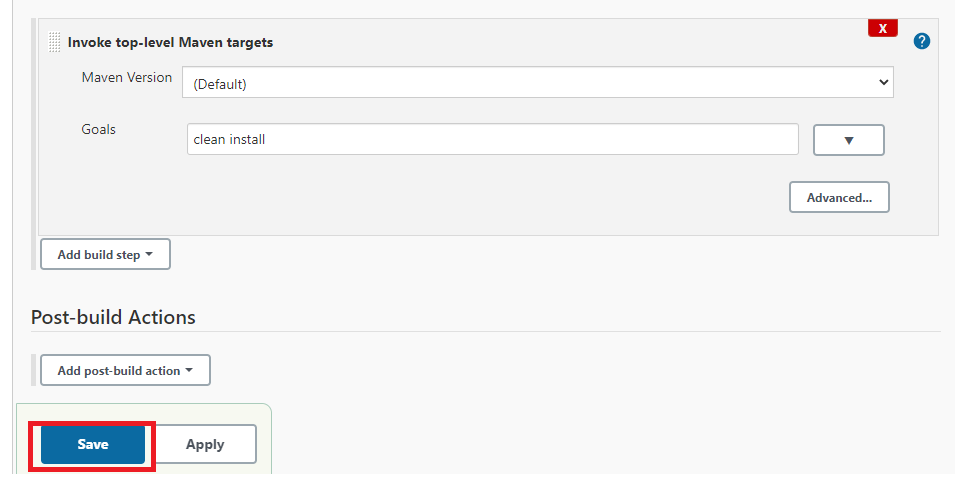
Choose build options according to the application.



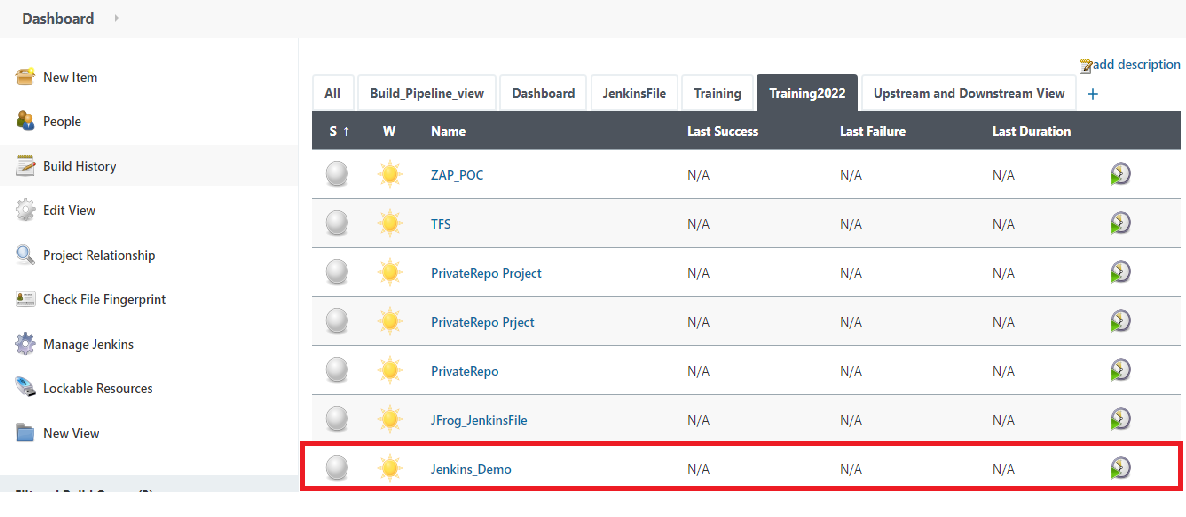
Provide Goals: **clean install**

****

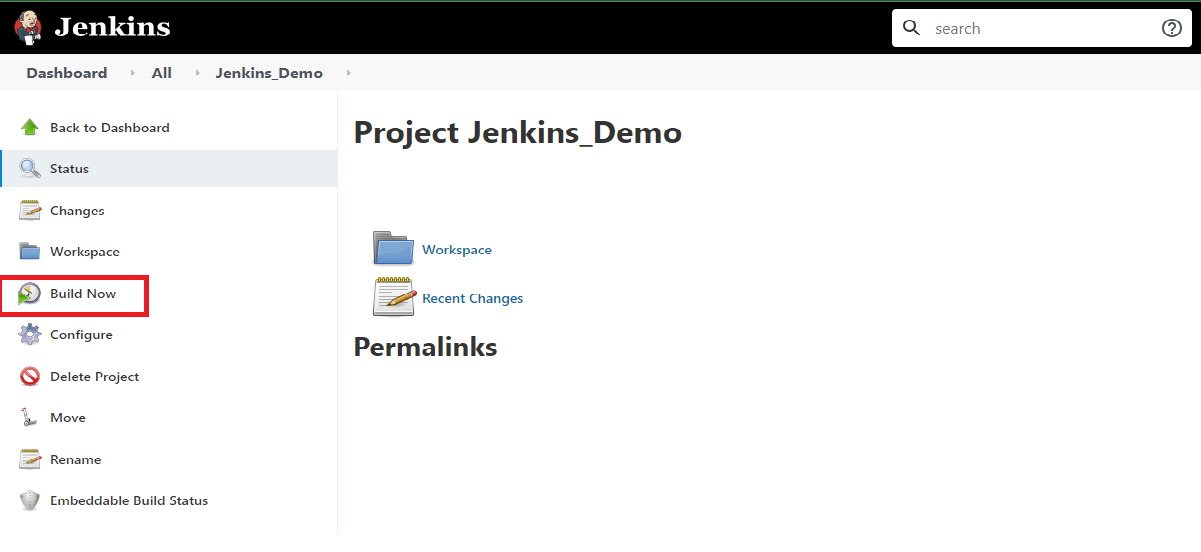
Finally click on **Save Button** to save the job configuration**.**

****

After saving the job configuration, you will be redirected to Jenkins Dashboard. Click on the job which you have created.



Now to build the Job, click on **Build Now**.



1. **CONFIGURATION GUIDELINES FOR JENKINS**

## Following are the guidelines for developers:

### Backup Jenkins home regularly

* The Backup plugin adds a new Backup manager item in the Manage Jenkins page.
* Backing Jenkins's configuration (using the Backup Jenkins configuration link)
* Restoring Jenkins's configuration from a previous backup (using the Restore Jenkins configuration link)

### Use fingerprinting to manage dependencies.

When you have interdependent projects on Jenkins, it often becomes hard to keep track of which version of this is used by which version of that. Jenkins supports "file fingerprinting" to simplify this, so make best use of it.

### The most reliable builds will be clean builds, which are built fully from source code control.

To ensure a build can be reproducible, the build must be a clean build, which is built fully from Source Code Control. This practice also implies that all code including third-party jars, build scripts, release notes, etc. must be checked into Source Code Control.

### Always configure job to generate trend reports and automated testing while running builds.

Trends help project managers and developers quickly visualize current project progress status. Moreover, unit testing is often not enough to provide confidence that the delivered software complies with the desired quality. The more you test the software, the better the delivered software complies with the desired quality.

### Set up Jenkins on the partition that has the maximum free disk-space

Jenkins needs some disk space to perform builds and keep archives. All the settings, build logs, artifact archives are stored under the JENKINS\_HOME directory. Simply archive this directory to make a backup. Similarly, restoring the data is just replacing the contents of the JENKINS\_HOME directory from a backup.

### Setup different job for each maintenance or development branch we create.

One of advantages of using CI tools is to detect problems early in the development lifecycle. Setting up a different job/project for each branch you create will help to maximize the benefit of detecting problems early as part of supporting parallel development efforts and reducing risk.

### Allocate different ports for parallel project builds and avoid scheduling all jobs to start at same time.

Multiple jobs running at the same time often cause collisions. Try to avoid scheduling all jobs to start at the same time. Allocate a different port for parallel project builds to avoid build collisions.

## Following are the guidelines for Jenkins Admin/Lead:

### 4.2.1 Monitoring Jenkins.

* Regularly monitor Jenkins with different monitor tools available
* Monitoring with Datadog
* Monitoring with Prometheus and Grafana
* Monitoring with JavaMelody
* Other Monitoring Plugins

### 4.2.2 Securing Jenkins.

* Access control, which ensures users are authenticated when accessing Jenkins and their activities are authorized.
* Protecting Jenkins against external threats.
* You should lock down the access to Jenkins UI so that users are authenticated, and appropriate set of permissions are given to them. This setting is controlled mainly by two axes:
  1. Security Realm, which determines users and their passwords, as well as what groups the users belong to.
  2. Authorization Strategy, which determines who has access to what.

### 4.2.3 Viewing & keeping logs.

* Jenkins uses *java.util.logging* for logging. The *java.util.logging* system by default sends every log above INFO to STDOUT.
* Jenkins is equipped with a GUI for configuring/collecting/reporting log records of your choosing.
* Admin should maintain/version the log files for maintaining proper working & history of working.

### 4.2.4 Authenticating scripted clients

* To make scripted clients (such as WGET) invoke operations that require authorization (such as scheduling a build), use HTTP BASIC authentication to specify the username and the API token.
* Specifying the real password is still supported, but it is not recommended because the risk of revealing password, and the human tendency to reuse the same password in different places.

### 4.2.5 Integrate Jenkins with issue tracking system like JIRA to reduce the need for maintaining

### change log

* The integration helps to track changes as they are made, including build status, what build has been performed for this requirement or defects, and the link to the actual build results and artifacts.
* JIRA plugin is used to integrate Atlassian JIRA to Jenkins.

### 4.2.6 Archive unused jobs before removing them.

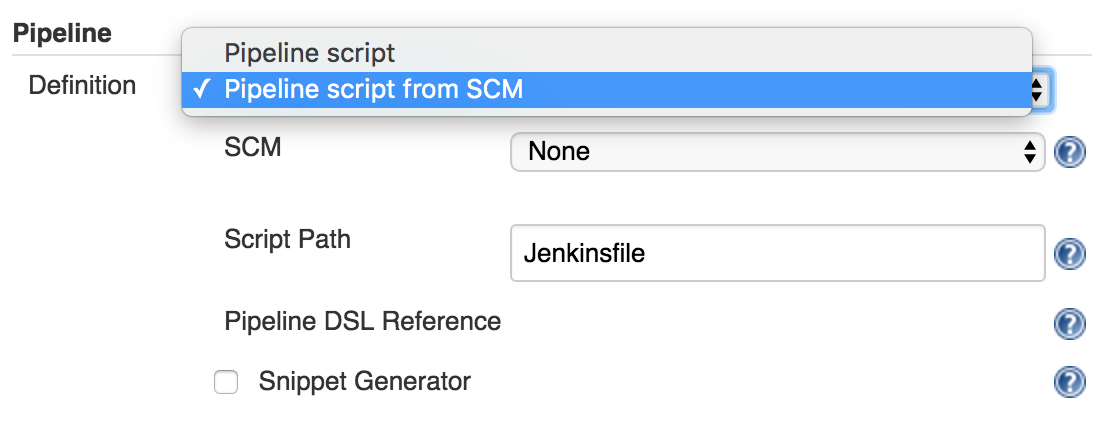
* All unused jobs should be archived so they can be resurrected if the need arises.

### 4.2.7 Setup email notifications mapping to all developer in the project, so that everyone on the team has his pulse on project’s current status.

* Configure each person on the people list with his or her correct email address and what role he or she is currently playing.
* Email-Ext plugin is used to configure every aspect of email notification. You can customize when an email is sent, who should receive it, and what the email says.

**ine**

* 1. **Following are the best practices for Jenkins Pipeline:**
     1. **Use the Jenkins Community Contributed Pipeline Plugin**
* Don’t use older plugins like Build Pipeline plugin or Buildflow plugin. Instead, use the Jenkins Community contributed Pipeline suite of plugins.
  + 1. **Develop Your Pipeline as Code**
* Use the feature to store your Jenkinsfile in SCM then version and test it like you do other software.
* Treating your pipeline as code enforces good discipline and also opens up a new world of features and capabilities like multi-branch, pull request detection and organization scanning for GitHub and BitBucket.



* You should also call your Pipeline script the default name: Jenkinsfile and start the following script header, so your IDE, GitHub and other tooling recognize it as Groovy and enable code highlighting: **#!groovy**.

### Do all work within a stage

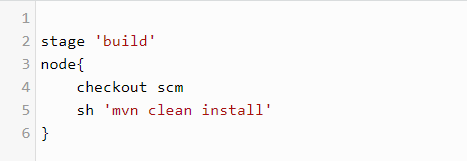
* Any non-setup work within your pipeline should occur within a stage block.
* Separating work into stages allows separating your pipeline into distinct segments of work.
* The Pipeline Stage View plugin visualizes stages as unique segments of the pipeline:



### All tasks & stages should be done in Nodes only

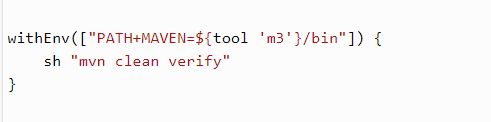
* Any material work within a pipeline should occur within a node block.
* By default, the Jenkinsfile script itself runs on the Jenkins master, using a lightweight executor expected to use very few resources.
* Any material work, like cloning code from a Git server or compiling a Java application, should leverage Jenkins distributed builds capability and run an agent node.

Example:



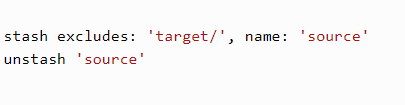
### Use withEnv instead of env global variable for setting variables

* While you can edit some settings in the env global variable, you should use the withEnv syntax instead.
* The env variable is global, changing it directly is discouraged as it changes the environment globally, so the withEnv syntax is recommended.
* Example:



### Prefer stashing files to archive them

* Before the stash capability was added to Pipeline DSL, archives were the best way to share files between nodes or stages in a pipeline.
* If you just need to share files between stages and nodes of your pipeline, you should use stash/unstash instead of archive.
* Stash and unstash are designed for sharing files, for example your application’s source code, between stages and nodes.
* Archives, on the other hand, are designed for longer term file storage (e.g., intermediate binaries from your builds).
* **Example:**



# INTEGRATION

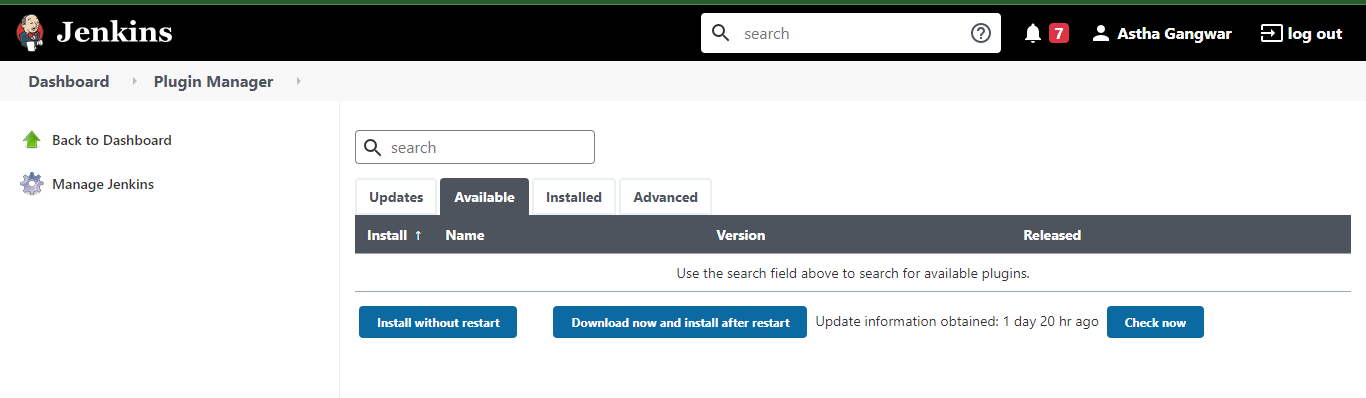
## Integration with SonarQube

As set-up of SonarQube and Jenkins server is already in place, this document will walk you through the different configurations we need to do on the Jenkins server to integrate SonarQube server with it.

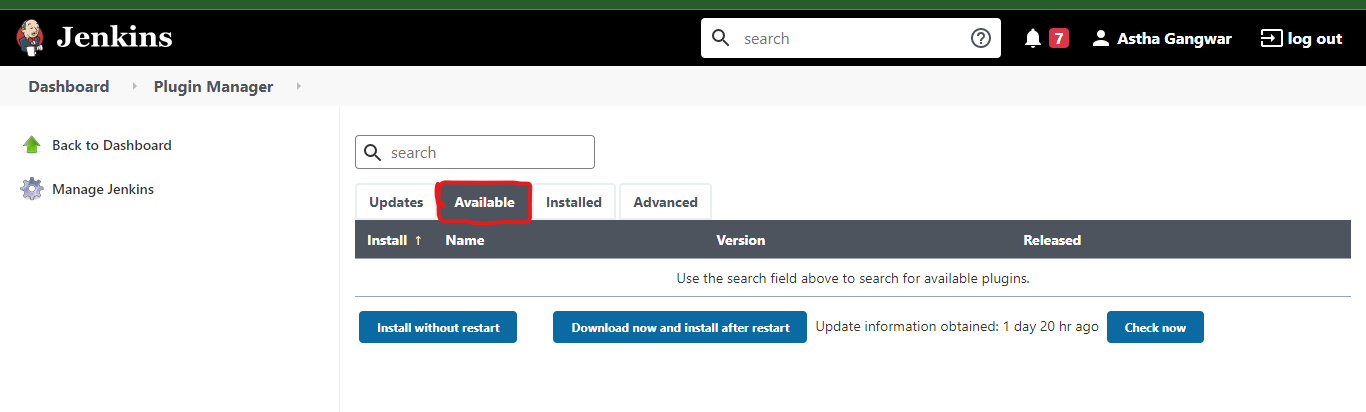
**Steps of Configuration in Jenkins:**

* **Installation of SonarQube Scanner Plugin:**

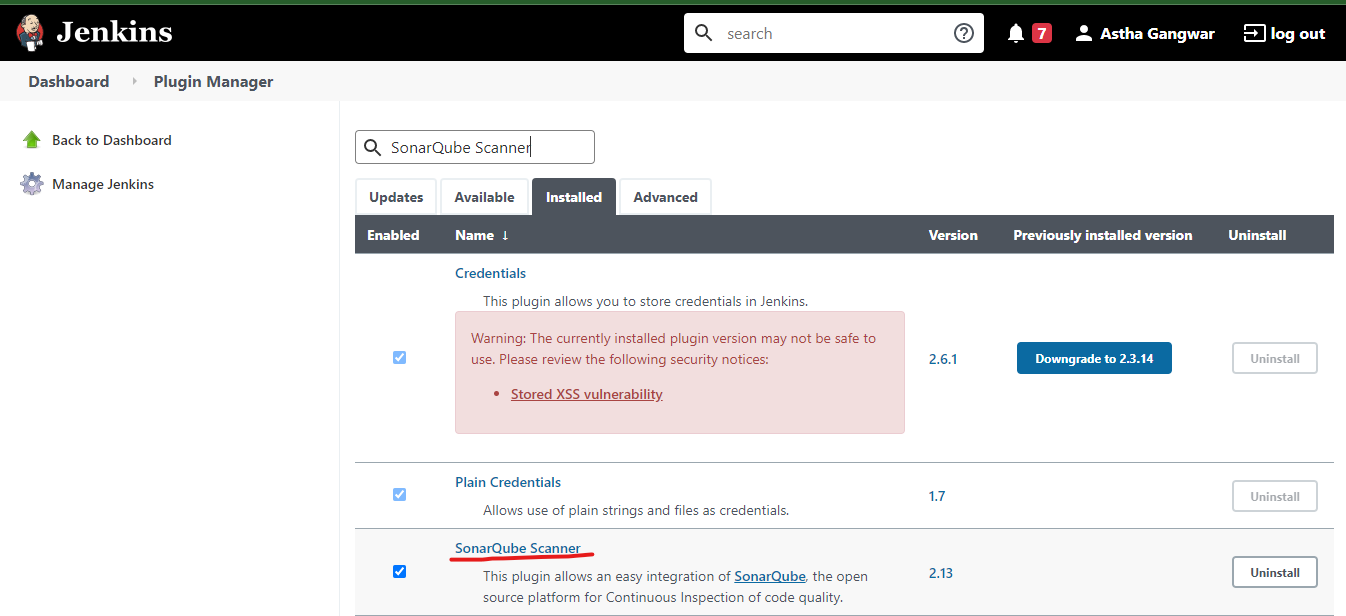
1. Log into Jenkins server as administrator.
2. Go to **Manage Jenkins => Manage Plugins**

****

1. Select **Available** tab



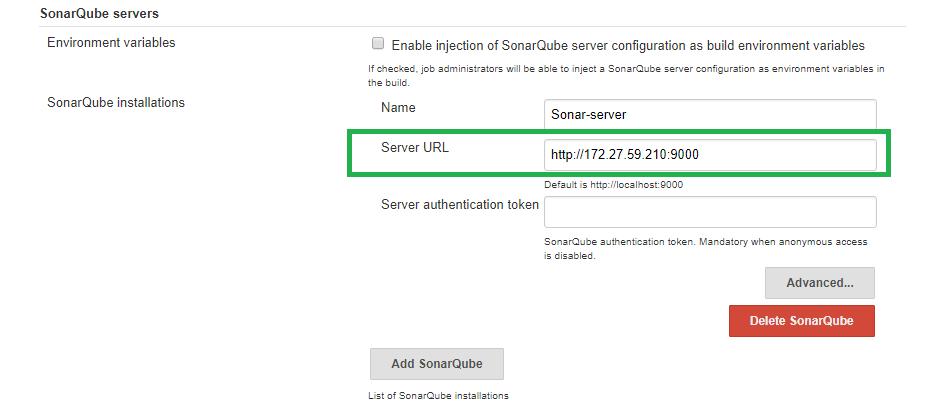
1. Type for **Sonar** plugin
2. You will be able to see a plugin called **SonarQube Scanner**



1. **Select** the plugin and click on **Install Without restart.**
2. Once installation is done, **Re-start** the Jenkins server.

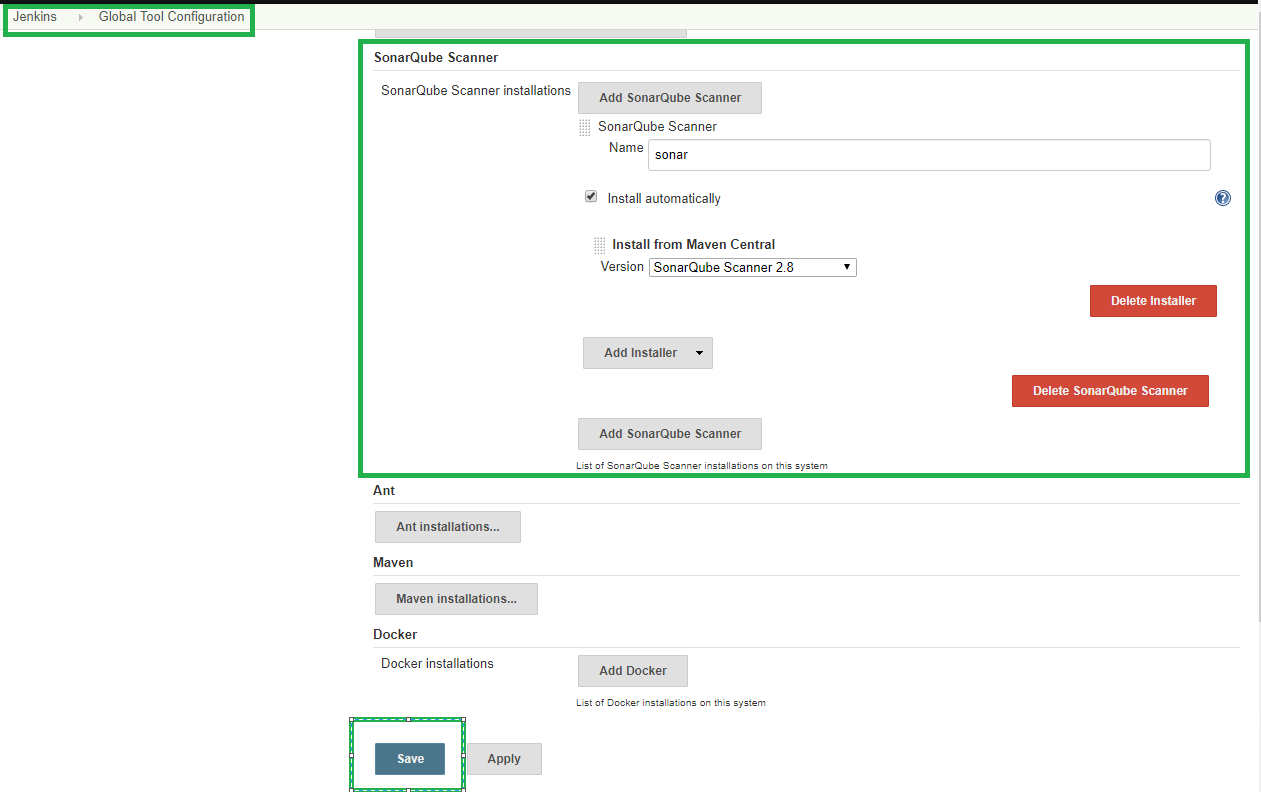
* **Configure SonarQube server on Jenkins:**

1. Log into Jenkins as an administrator and go to **Manage Jenkins > Configure System**
2. Scroll down to the **SonarQube configuration section**, click on **Add SonarQube**, and add the values you're prompted for.

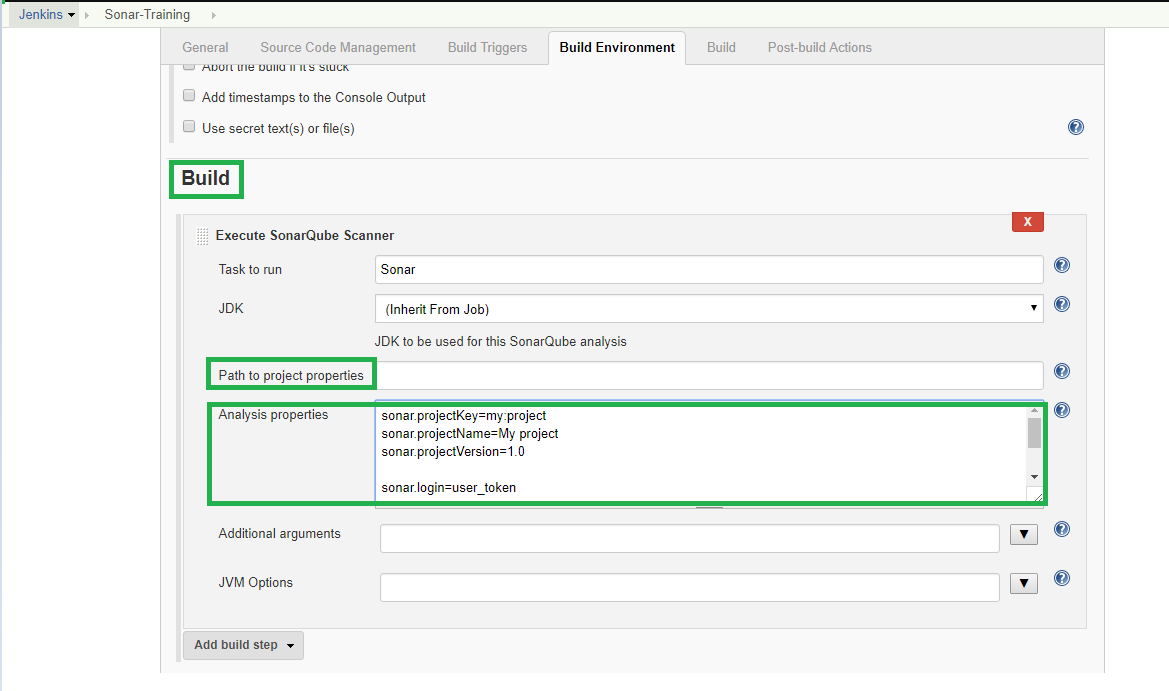


1. And finally click on **Save**.

* **Configure SonarQube scanner on Jenkins:**
  1. Log into Jenkins as an administrator and go to **Manage Jenkins > Global Tool Configurations.**
  2. Scroll down to the **SonarQube Scanner configuration** section and click on **Add SonarQube Scanner**. It is based on the typical Jenkins tool auto-installation. You can either choose to point to an already installed version of SonarQube Scanner (**uncheck** **'Install automatically'**) or tell Jenkins to grab the installer from a remote location **(check** **'Install automatically'**)

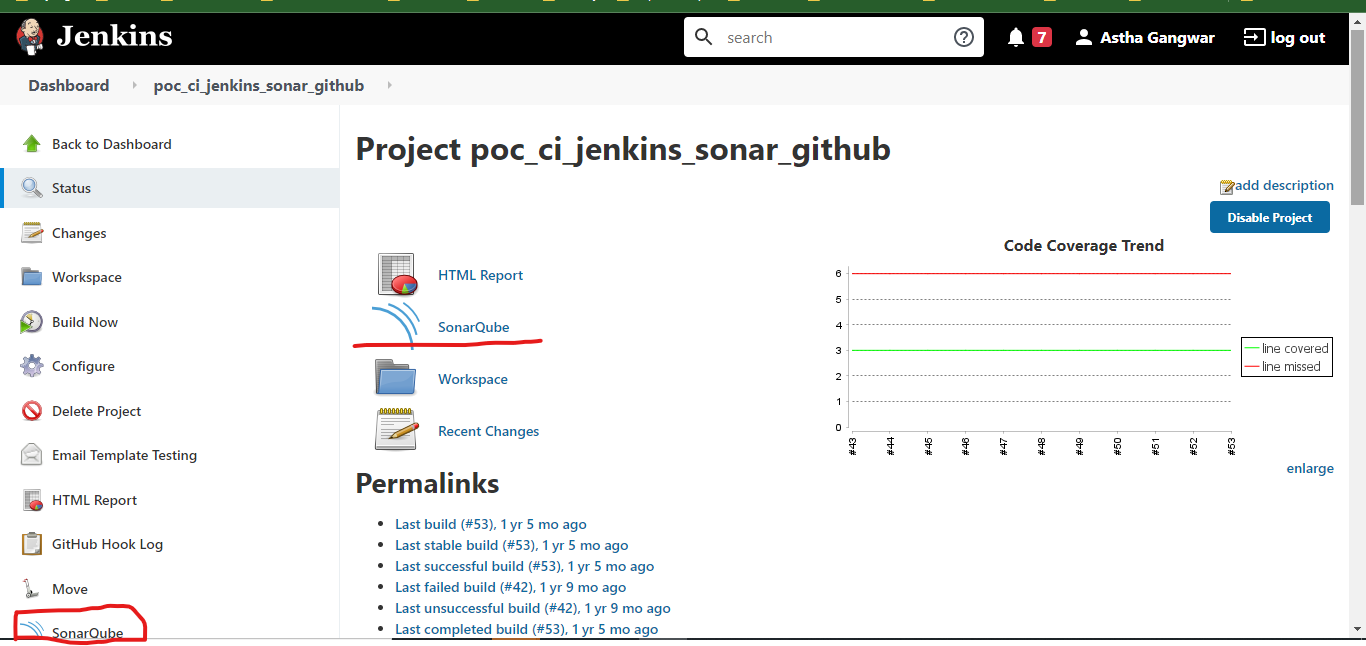


* **Configure SonarQube scanner on Jenkins:**
  1. **Configure** the project, and scroll down to the **Build** section.
  2. Add the SonarQube Scanner build step to your build*.*



* 1. Configure the SonarQube analysis properties. You can either **point to an existing sonar-project. properties file** or set the **analysis properties** directly in the Analysispropertiesfield
* **Browsing Result of SonarQube analysis on Jenkins:**

Run the build and once Build is successful, check for the result on Jenkins Job configuration Dashboard*.*



So, these are the basic steps for implementing integration between SonarQube and Jenkins.

## Integration with Email

To send mail through Jenkins File using pipeline Job of Jenkins.

**1. Pre-requisite:**

* Jenkins should be on public IP

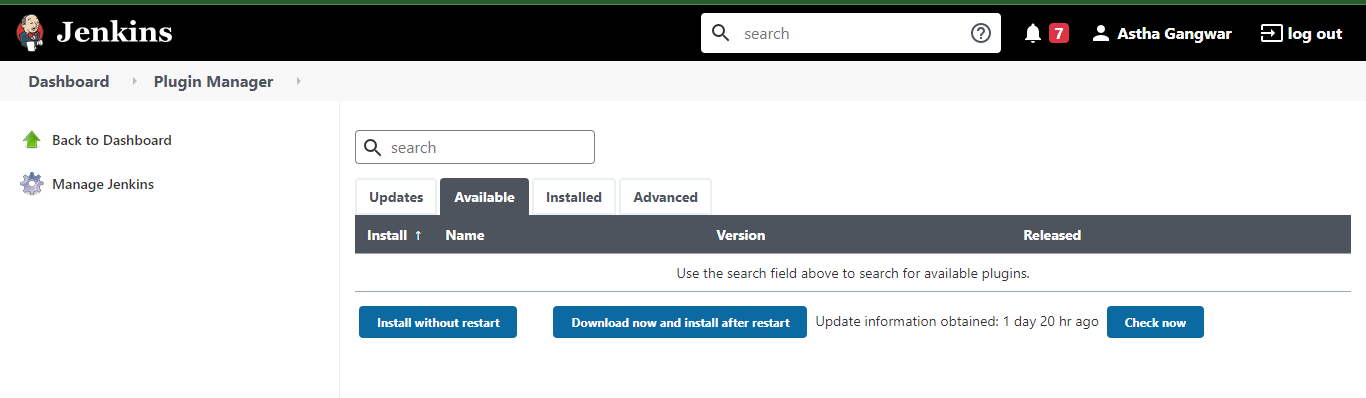
**2. Steps of Configuration in Jenkins:**

**Note:** This plugin comes with default installation plugin; in case you have not choose default installation of plugins then you have to install it manually.

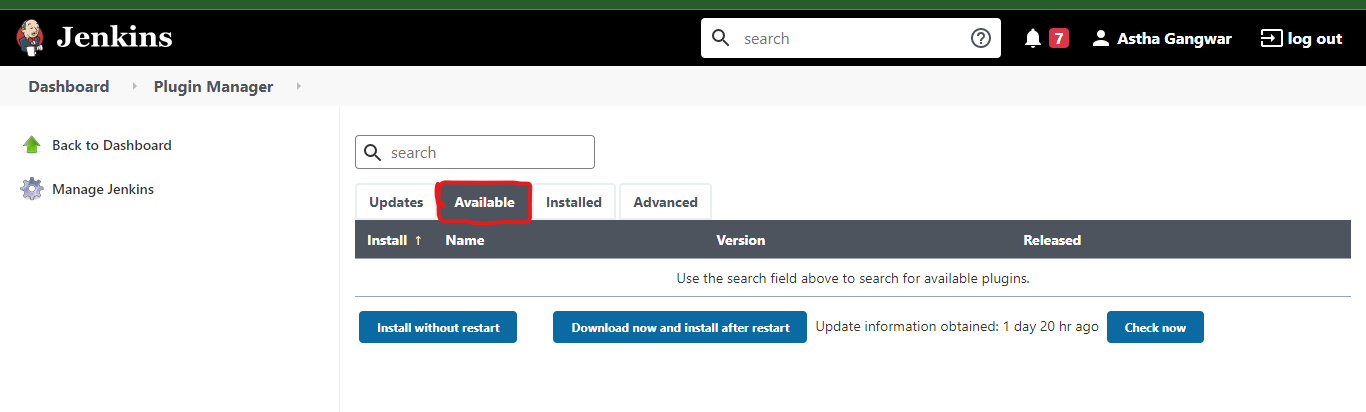
**1. Installation of Email Extension Plugin:**

1. Log into Jenkins server as administrator.
2. Go to:

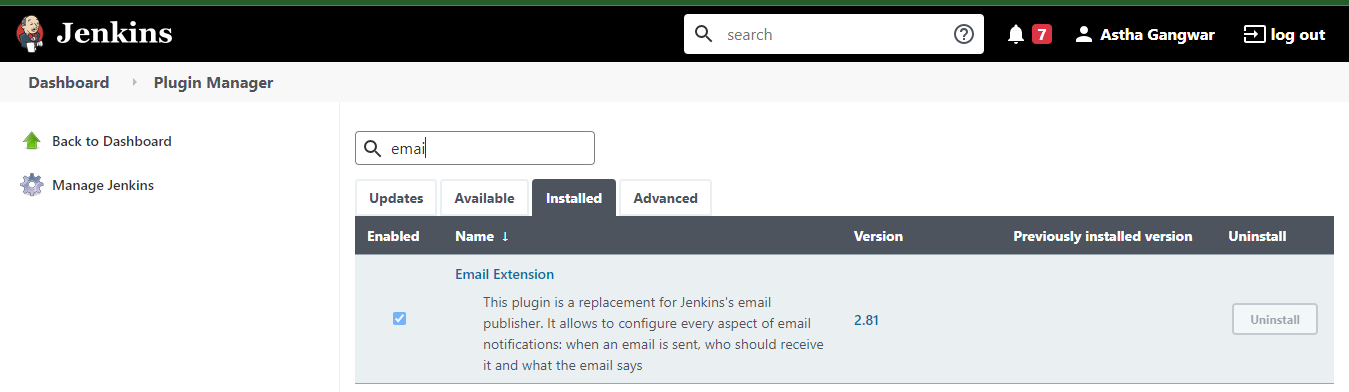
* **Manage Jenkins => Manage Plugins**

****

* Select **Available** tab



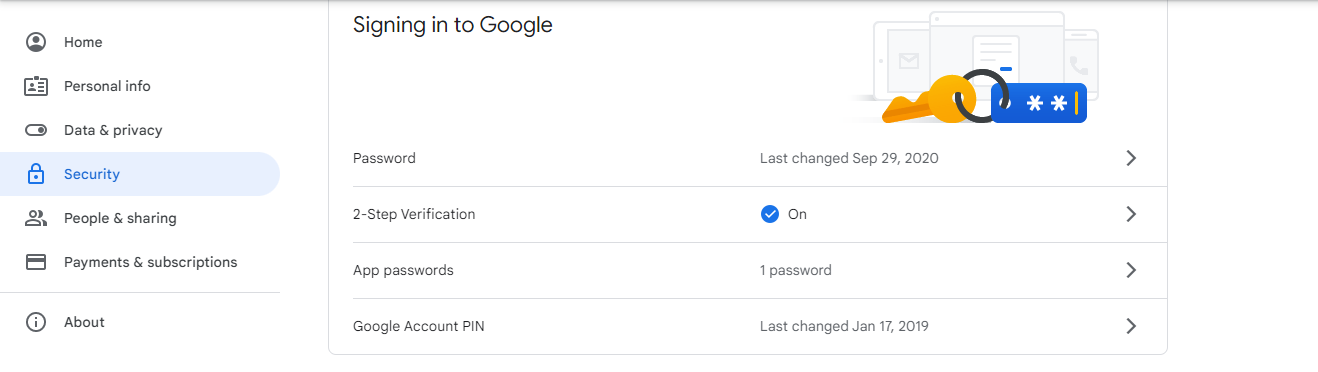
* Type for **Email Extension** plugin
* You will be able to see a plugin called **Email Extension**

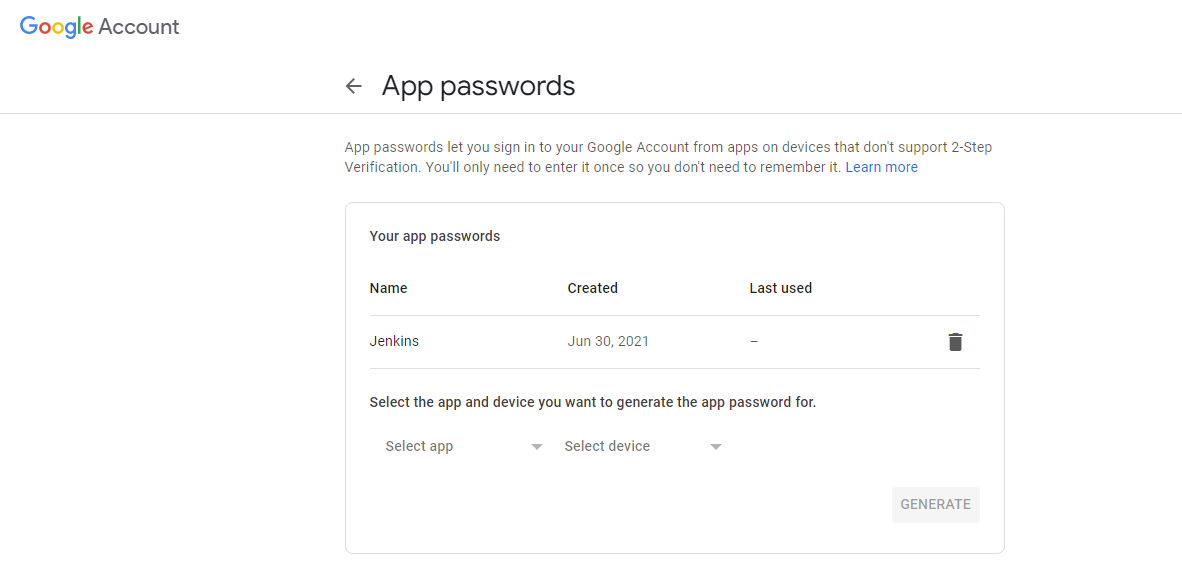


* **Select** the plugin and click on **Install Without restart.**
* Once installation is done, **Re-start** the Jenkins server.

1. **Change in Gmail/or any other SMTP server**

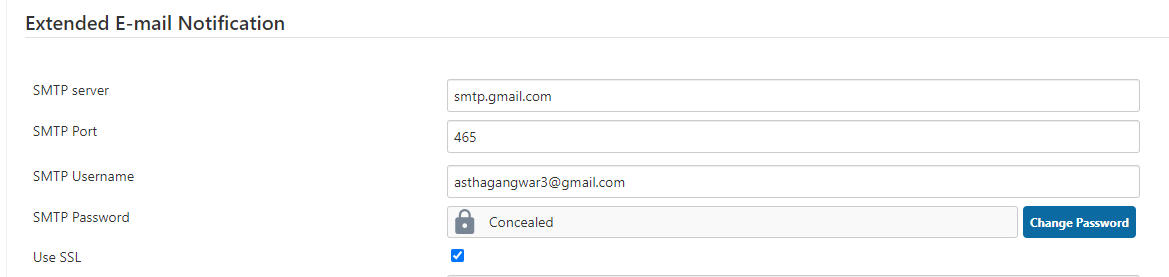
* Go to security settings of your account.
* Create app password.

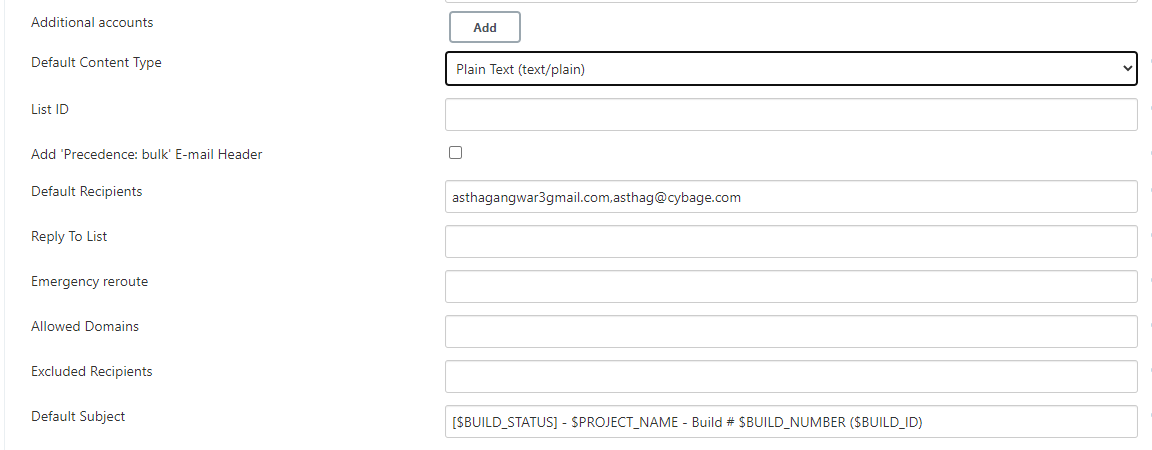




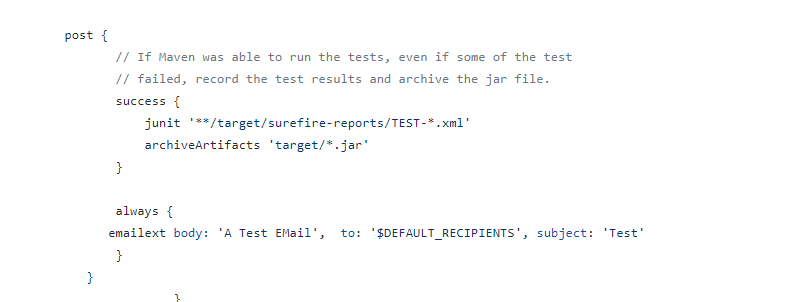
**Password created for app in Gmail, use that password while configuring Email Extension in SMTP password.**

1. **Configure properties in Email extension**

****



1. **Jenkinsfile Configuration**

****

## Integration with Slack

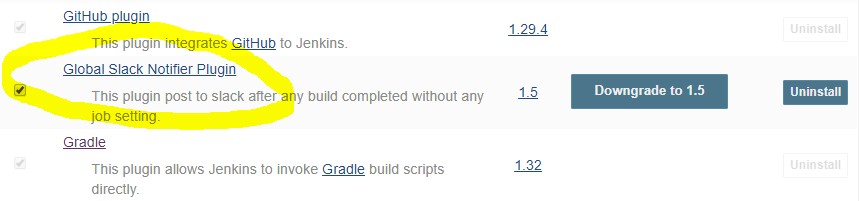
1. Create a new JOB in Jenkins, e.g., Slack Job

2. In configuration of the project, give the SCM and other pre-build steps you want.

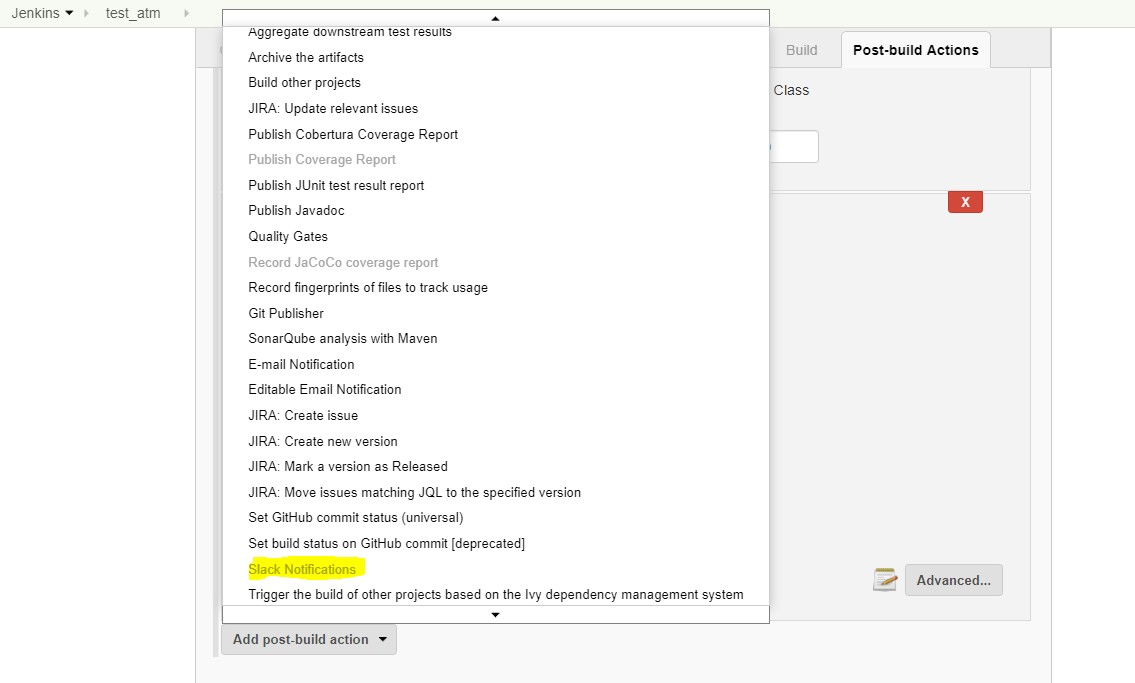
Graphical user interface, application

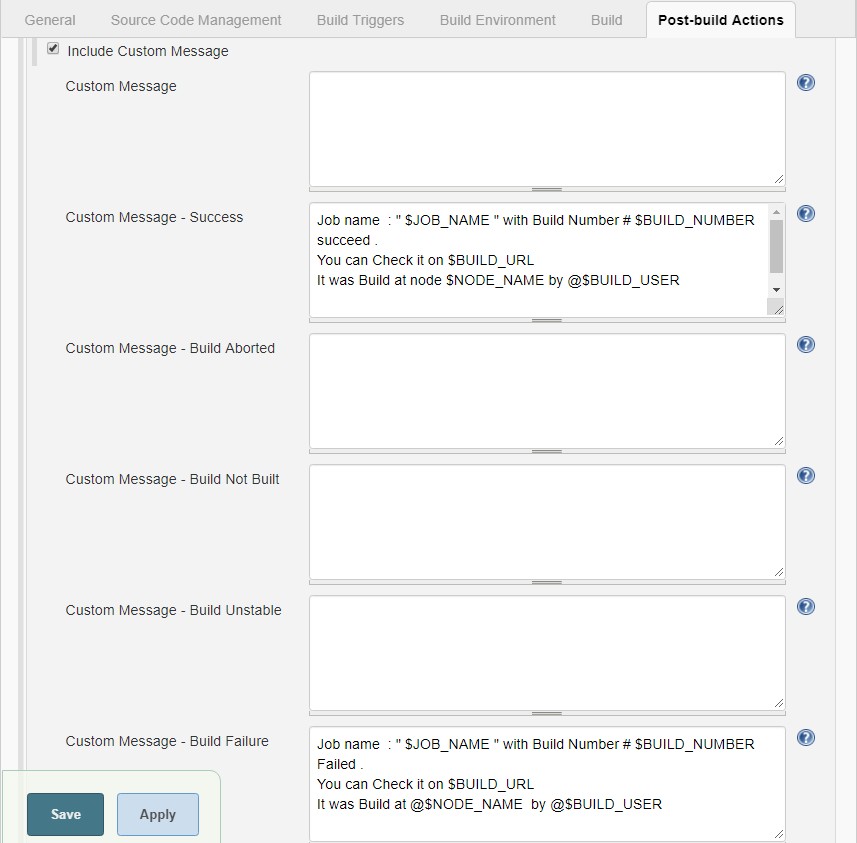
Description automatically generated

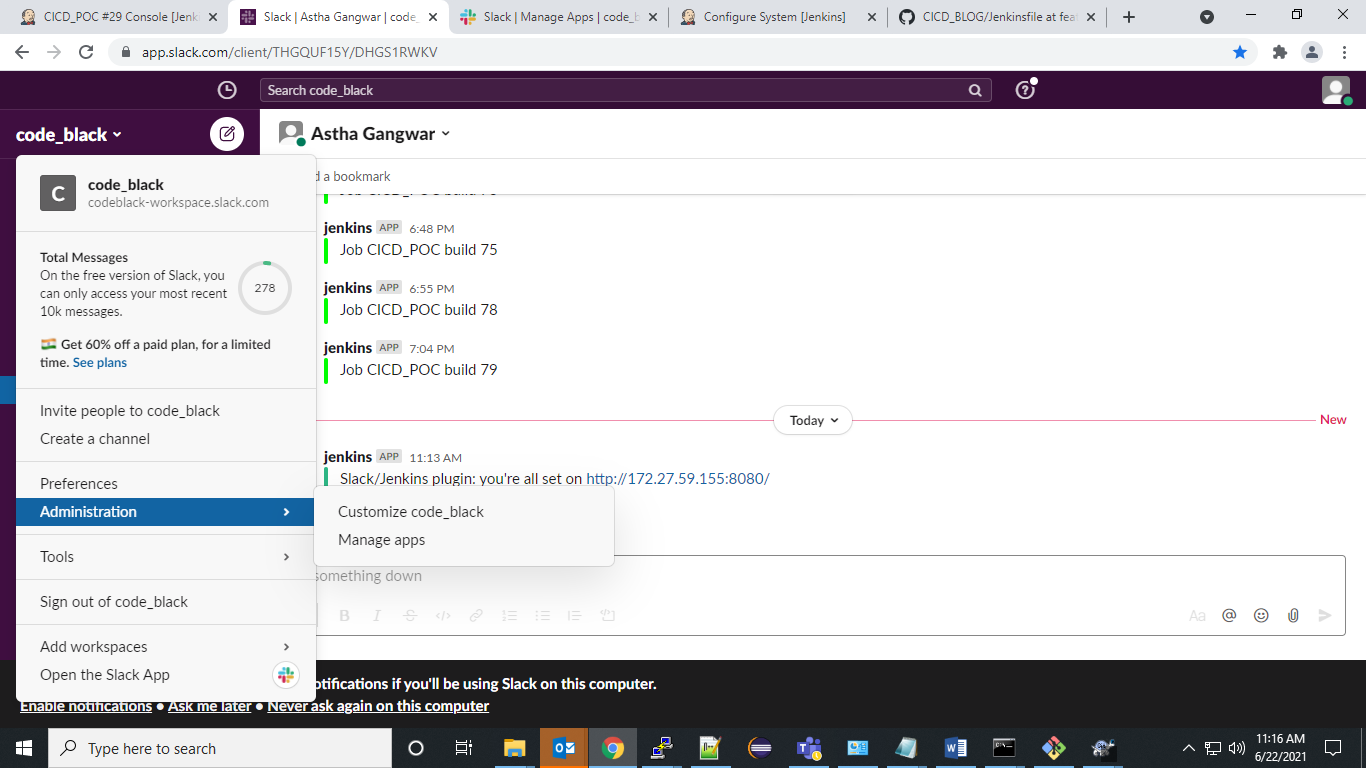
1. Make sure you have installed the following plugin in Jenkins:



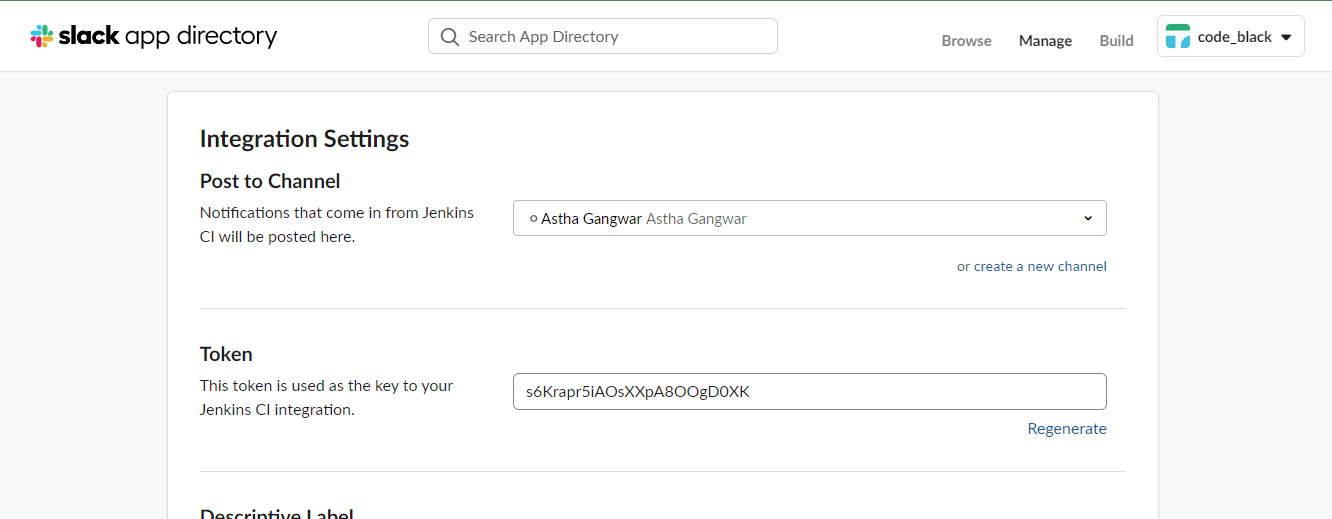
1. In configuration of job, go to post-build actions:

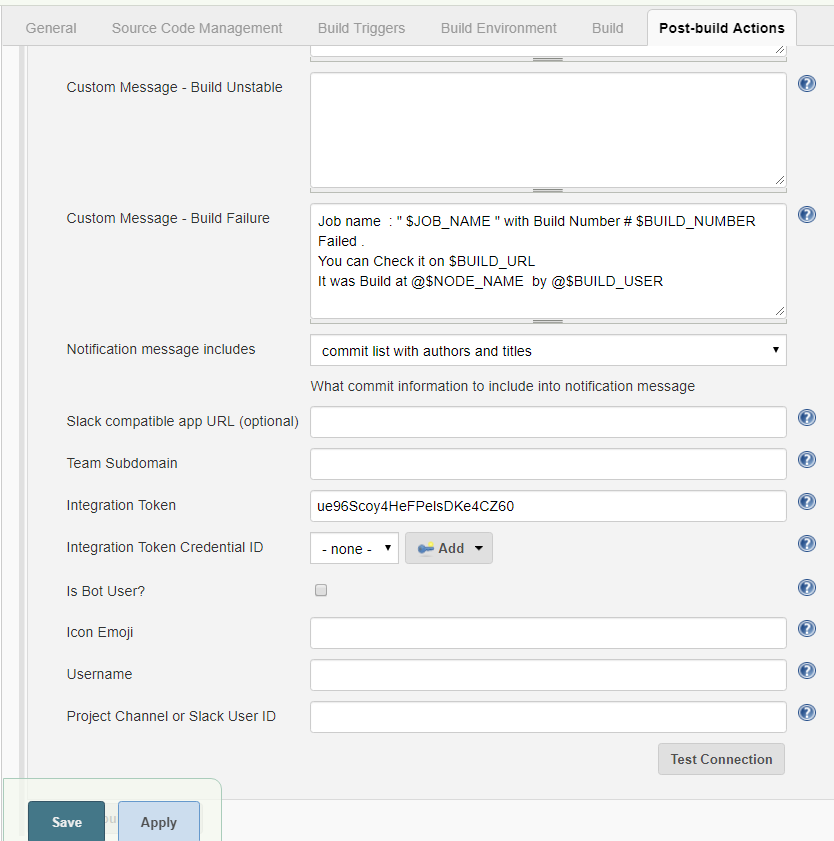


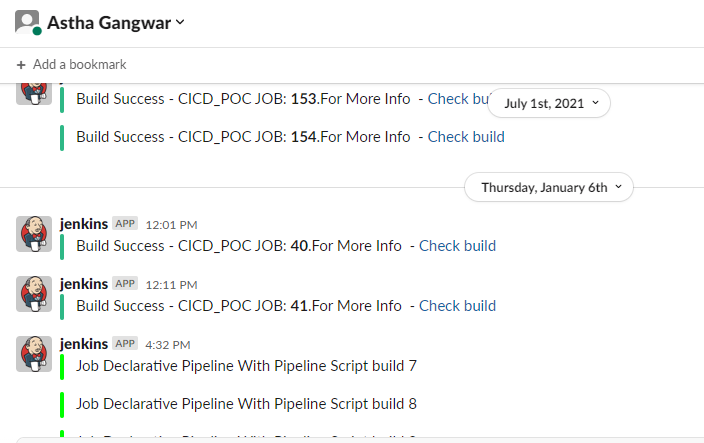
1. Select the notifications type you want and go to custom messages input box section below in advanced:
2. Go to slack and in administration tab and then go to manage apps.



1. Go to Slack and create integration token for Jenkins CI from add apps section in slack and Select “Add Jenkins CI Integration”.
2. After adding Jenkins CI Integration, token is generated automatically:



1. Copy the Token key and paste it in configuration of Jenkins job: SlackJob
2. To check whether the plugin integration of slack is working, built that job and watch out for any notification in slack:



# NAMING CONVENTION TO BE FOLLOWED IN JENKINS

* Naming convention for folders should be in the form:

**{company-name}-{project-name}-{application-name}-{entity}**

**E.g. cybage-projectxyz-middleware-folder**

* Naming convention for jobs in Jenkins should be in form:

**{company-name}-{project-name}-{environment}-{application-name}-{task}-{entity}**

**E.g. cybage-alm-dev-ems-deploy-job**

* Also while mentioning commit message in git, mention the jira issue-id in the beginning of message:

Commit message: “**JRA-08 Updated login page**” . Where *JRA-08* is jira issue id & the rest is commit message.

* Naming convention for Slave name in Jenkins should be in the form:

**{company-name}-{project-name}-{os}-{environment}-{entity}**

**{company-name}-{project-name}-{os}-{environment}-{application-name}-{task}-{entity}**

* Naming convention for plugin specific environment location path name:

**{tool-name}-{tool-version}**

**{slave-initials}-{tool-name}-{tool-version}**

**E.g., maven-3.6,**

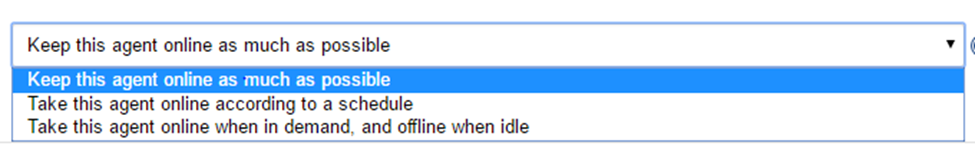
**almslave-maven-3.4**

# BUILD CONFIGURATION FOR MASTER/SLAVE ARCHITECTURE

Jenkins supports the "master/slave" mode, where the workload of building projects is delegated to multiple "slave" nodes, allowing a single Jenkins installation to host a large number of projects, or to provide different environments needed for builds/test.

Below are the requirements for the machine to be agent.

* Agents are typically generic x86 machines with enough memory to run specific build types. The agent’s configuration depends on the builds it will be used for and on the tools required by the same builds.
* Configuring a machine to act as an agent inside your infrastructure can be tedious and time consuming. This is especially true when the same set-up has to be replicated on a large pool of agents. Because of this, is ideal to have fungible agents, which are agents that are easily replaceable. Agents should be generic for all builds rather customized for a specific job or a set of jobs. The more generic the agents, the more easily they are interchanged, which in turn allows for a better use of resources and a reduced impact on productivity if some agents suffer an outage.
* Here are the best practices suggested for better use of distributed build architecture of Jenkins as Continuous Integration Server.
* For Windows,
  1. Ensure that java version of slave machine is compatible with master.
  2. JNLP connector must be up and online so that master machine can connect with the service
  3. Tool’s location for slave should be provided and configured for running distributed builds.
  4. If Jenkins slave installed as a service, ensure that service is always up and running.
  5. Enough no of build executors must be supplied.
* For Linux,
  1. Ensure that java version of slave machine is compatible with master.
  2. Master machine must have SSH access (through keygen or username/password) and Slave.jar copied on remote directory (after connection through SSH tunnel) should not be deleted.
  3. Tool’s location for slave should be provided and configured for running distributed builds.
  4. Enough number of executers must be supplied.
  5. Proper scheduling strategy need to be checked while configuring new node as slave among the following:



# REFERENCE LINKS

<https://www.jenkins.io/doc/>

<https://www.jenkins.io/doc/book/installing/>

<https://www.jenkins.io/doc/book/pipeline/pipeline-best-practices/>

<https://www.jenkins.io/doc/book/managing/>