accenturetechnology
Learning and Knowledge Management

JENKINS Tool Overview

Table of Contents

Module 2

Exercise 2.1: Install Jenkins and Access the Dashboard

Scenario

Perform activities to install Jenkins and to get started with tasks

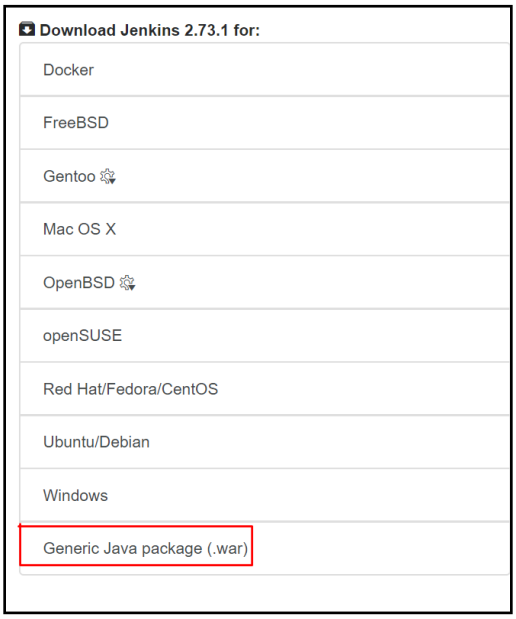
Prerequisite: Java 8, 512 MB RAM

Walkthrough

1. Download and install the war file
2. Set the initial admin password
3. Install suggested plugins
4. Create first admin user and start accessing the Jenkins dashboard

Steps

1. Download and install the war file

1	a. Navigate to https://jenkins.io/download/ b. Download **Generic Java package (.war)** and save it into your local disk. Example: **C:\Softwares** 
2	Open Command prompt and set the JENKINS_HOME directory by executing the below command: **C:\Softwares>set JENKINS_HOME=C:\Jenkins**
3	Install Jenkins in the port 8082 by executing the command as highlighted in the below screen **java -jar jenkins.war --httpPort=8082** Note: By default, Jenkins get started in the port number 8080.

The command prompt displays the following message, when Jenkins starts running successfully

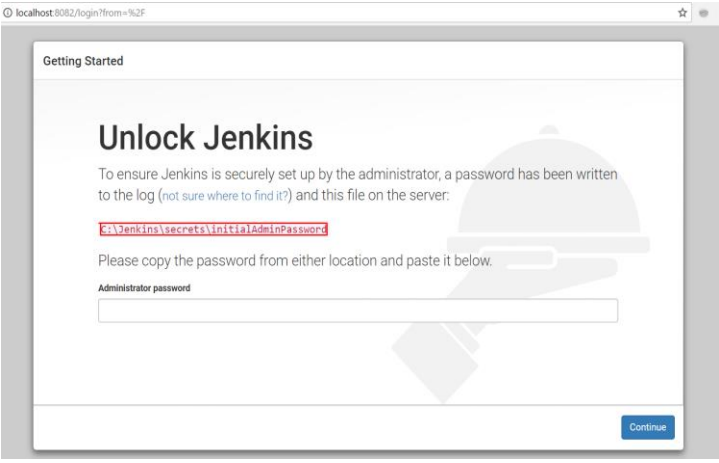
```
Oct 04, 2017 3:22:17 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Oct 04, 2017 3:22:18 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Oct 04, 2017 3:22:19 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Oct 04, 2017 3:22:19 PM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
Oct 04, 2017 3:22:21 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Oct 04, 2017 3:22:21 PM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 17,796 ms
```

2. Set the initial admin password

1

You can access Jenkins in the path: http://localhost:8082

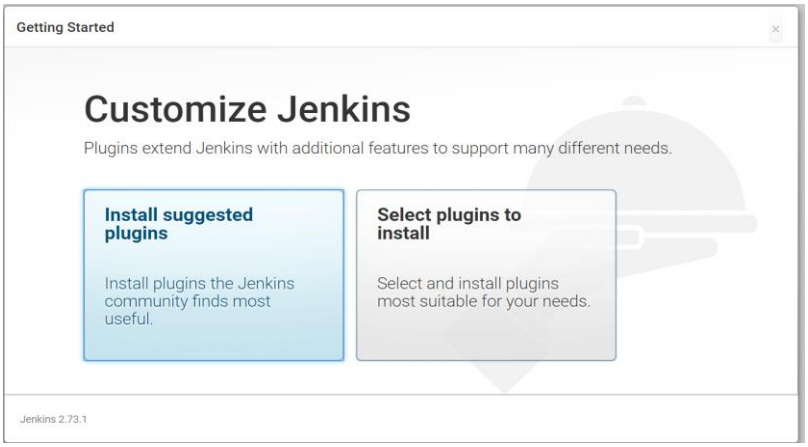
To start working with Jenkins dashboard, first set the administrator password as provided in the location: **C:/Jenkins\Secrets\initialAdminPassword**



2

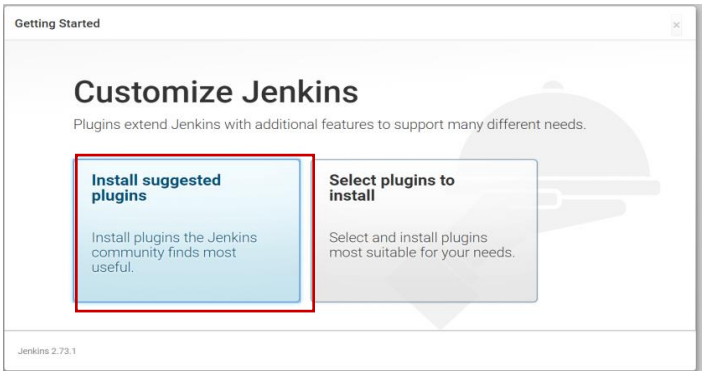
Copy the admin password provided in the location and paste it as shown below and click **Continue**

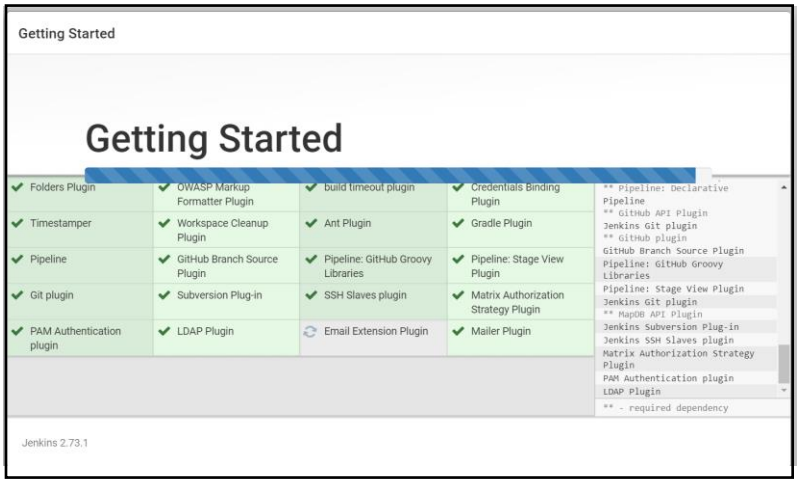
The Jenkins displays the following screen to install plugins before you start accessing the dashboard



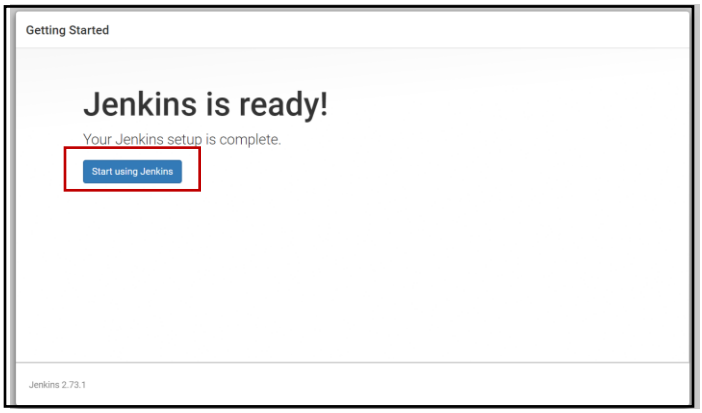
3. Install suggested plugins

1 Click **Install suggested plugins**



	The following screen appears to indicate the default plugins that are installed successfully: 

4. Create first admin user and start accessing the Jenkins dashboard

1	Enter the respective details for the fields, and click **Save and Finish** to complete Jenkins setup 
2	Click **Start Using Jenkins**, when the setup is complete 

	You can now start using the Jenkins dashboard 

Exercise 2.2: Configure Jenkins

Scenario

Perform activities to configure Jenkins with JDK, Git, Maven and Ant

Prerequisite: Working knowledge of JDK, GIT, Maven and Ant

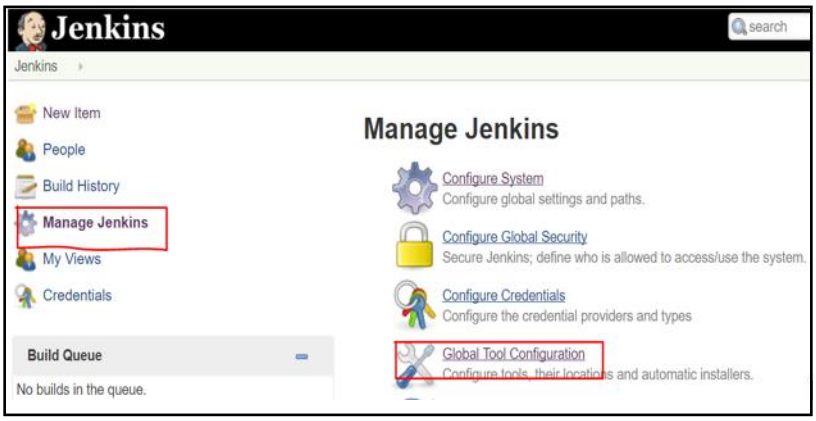
Note: The JDK, Git, Maven and Ant packages should be installed in your local system to configure with Jenkins

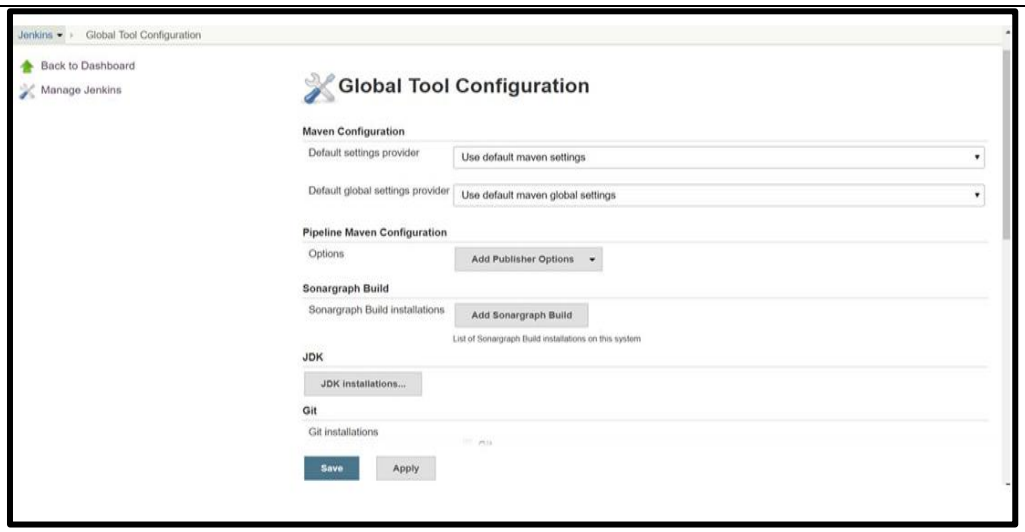
Walkthrough

1. Access Global Tool Configuration section
2. Configure Jenkins with JDK
3. Configure Jenkins with Git
4. Configure Jenkins with Maven
5. Configure Jenkins with Ant and save all the configurations

Steps

1. Access Global Tool Configuration section

1	Open the Jenkins dashboard in the URL http://localhost:8082
2	Navigate to **Manage Jenkins → Global Tool Configuration** option  This results in the **Global Tool Configuration** page, where you can choose the components to configure Jenkins with.

2. Configure Jenkins with JDK

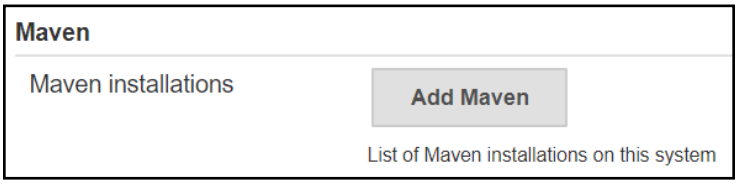
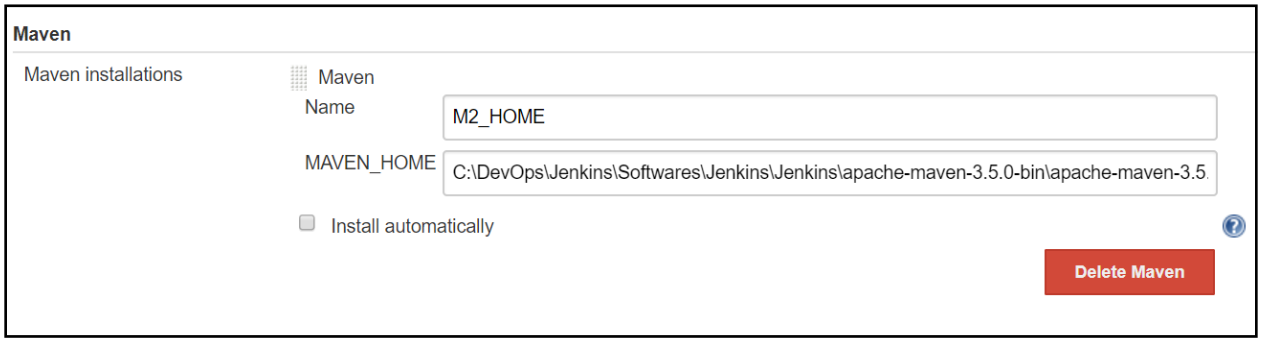
1	a. In the Global Tool Configuration page, navigate to the **JDK** section b. Click **Add JDK** 
2	a. Uncheck the **Install automatically** checkbox b. Specify the name and path of the JDK installed in the system in the JAVA_HOME field as shown below 

3. Configure Jenkins with Git

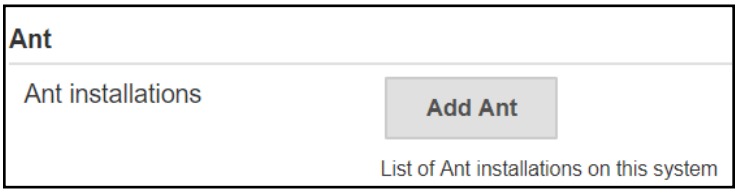
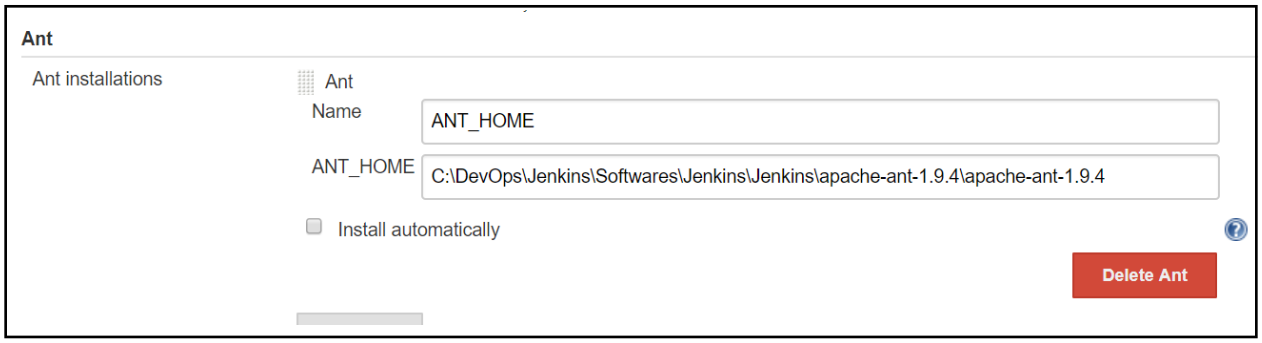
1	a. Navigate to the **Git** section b. Click **Add Git**

	
2	a. Uncheck the **Install automatically** checkbox b. Specify the name and path of the Git installed in the **Path to Git executable** field as shown below 

4. Configure Jenkins with Maven

1	a. Navigate to the **Maven** section b. Click **Add Maven** 
2	a. Uncheck the **Install automatically** checkbox b. Specify the name and path of the Maven installed in the **M2_HOME** field as shown below 

5. Configure Jenkins with Ant and save all the configurations

1	a. Navigate to the **Ant** section b. Click **Add Ant** 
2	a. Uncheck the **Install automatically** checkbox b. Specify the name and path of the Ant installed in the **ANT_HOME** field as shown below 
3	Click the **Save** button to save the JDK, Git, Maven and Ant configurations

Exercise 2.3: Install Plugins

Scenario

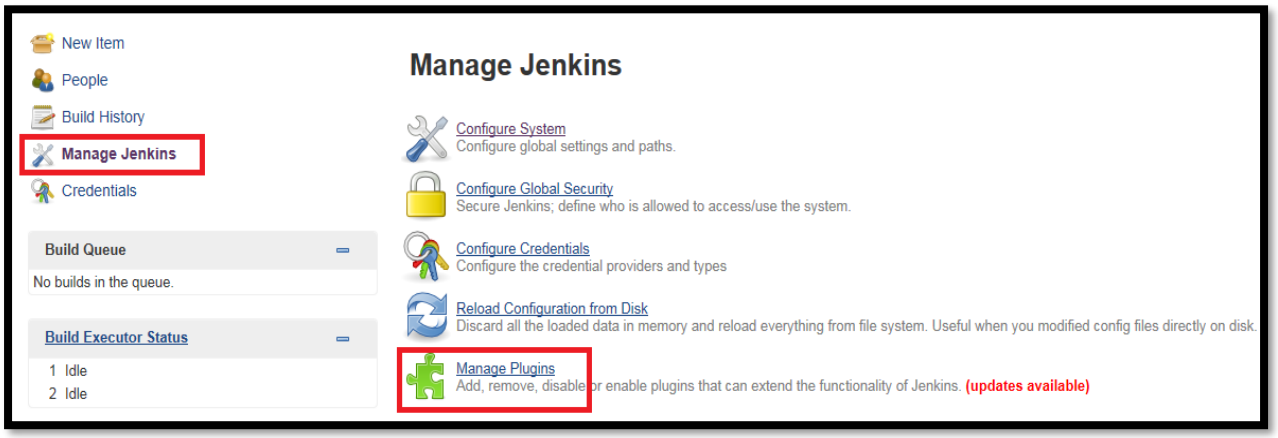
Perform activities to install plugins that facilitates to perform tasks in Jenkins.

Walkthrough

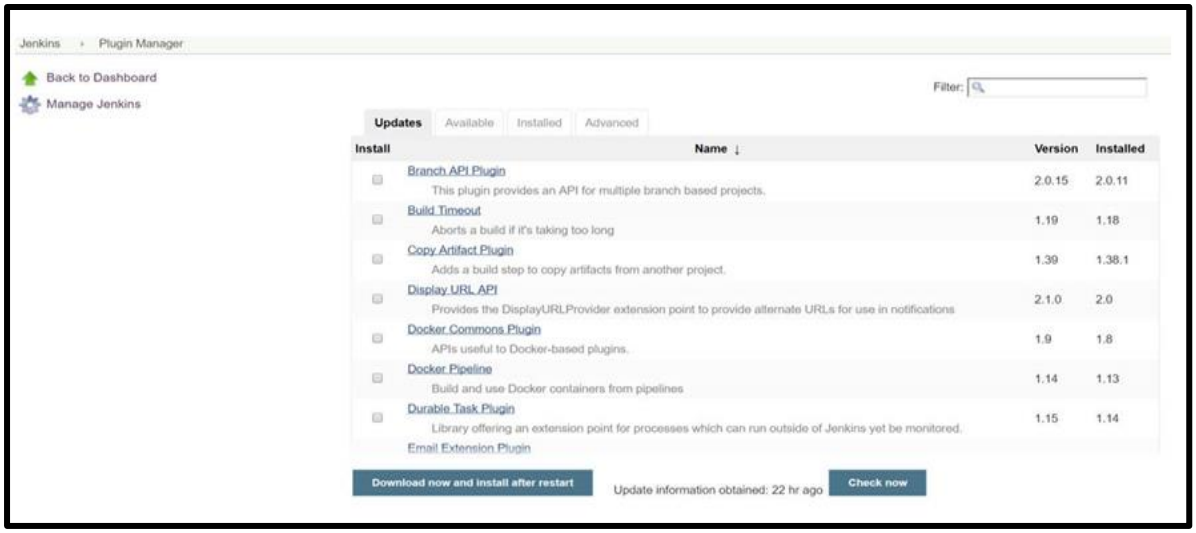
1. Search and install the required plugins

Steps:

1	In Jenkins dashboard, navigate to **Manage Jenkins → Manage Plugins** option.

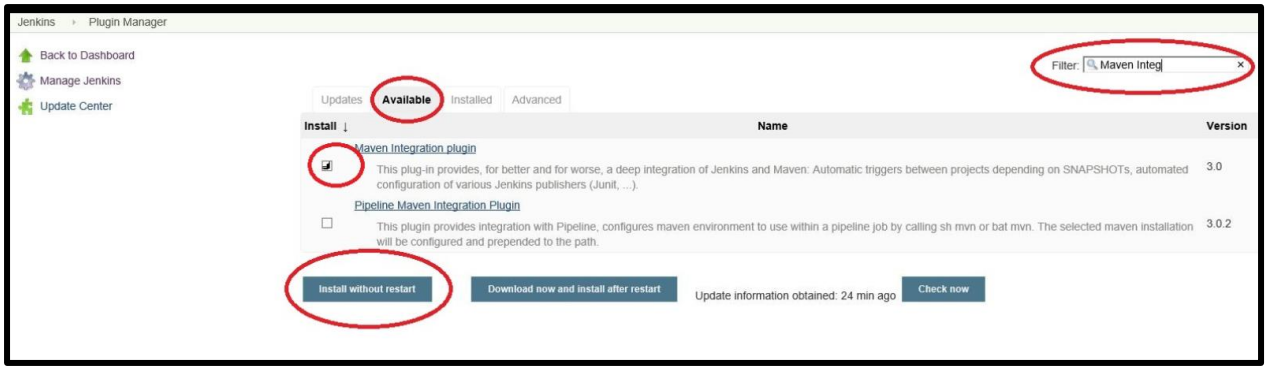


You will see the **Plugin Manager** page



2

a. In the Plugin Manager page, click the **Available** tab
b. Search for the required plugins
c. Check the box for the required plugins
d. Click the **Install without restart** tab



	Note: Select the plugins shown below to install: 
	All the selected plugins will be installed and ready to use in Jenkins **Note: We can install all plugins which are required for a project by following the above procedure.**

End of Module 2

Module 3

Exercise 3.1: Build Job

Scenario

Perform activities to build job in Jenkins.

Prerequisite:

1. Working knowledge of GIT and Maven
2. The project to build should be created in GitHub server (The steps are included below)
3. JDK, Git, Maven and Ant should be installed in the C Drive of your local system to perform this exercise
4. To perform exercise Git should be installed in the system
   Steps to be followed to install Git:
   a. Click on Git Download Git from the below URL
      i. https://git-for-windows.github.io/
   b. Start installation
   c. Select “Git bash” and “Git GUI”
   d. Select “Use Git from windows command prompt”
   e. Keep remaining options as default and click Install button.
   f. If Git installation is completed successfully, then follow these steps.
   g. Navigate and select Git Bash
5.

Note:

1. This exercise uses GitHub server to store source code and Maven to build project
2. The source code will be shared by faculty
3. The source code should be stored in C drive of the local system

Walkthrough

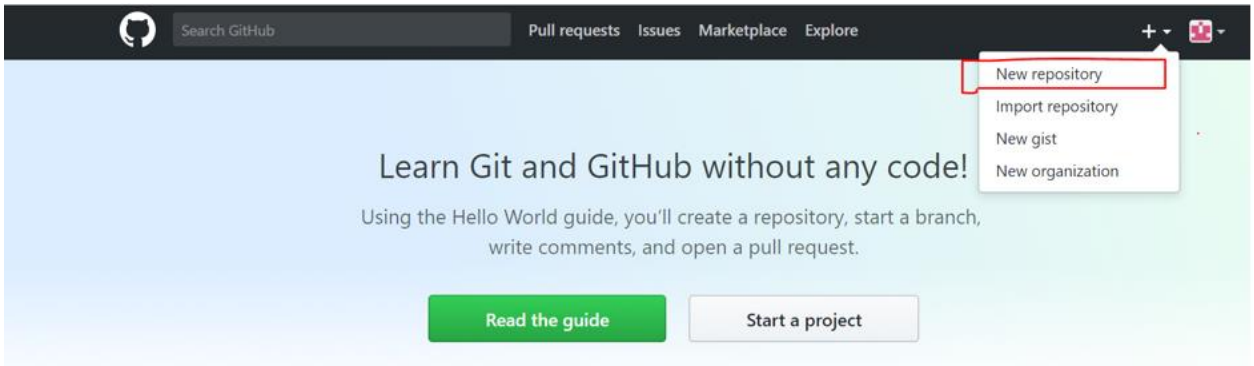
1. Install Git
2. Create project in GitHub server
3. Create job in Jenkins and configure it with the GitHub project
4. Configure Source Code Management with Git
5. Configure build job using Maven
6. Execute build
7. View build results

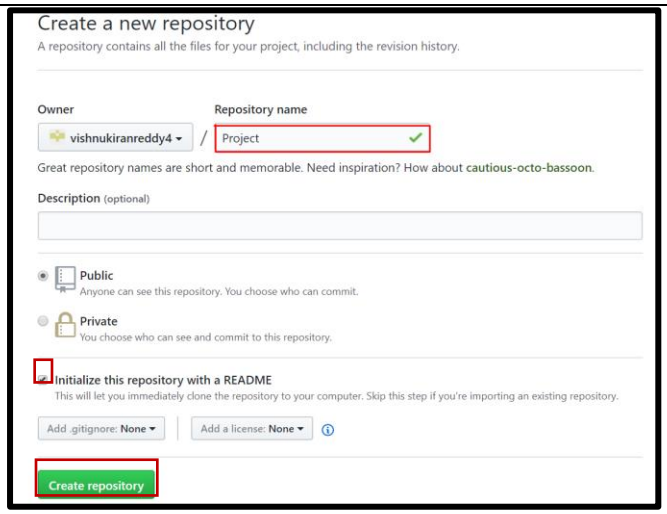
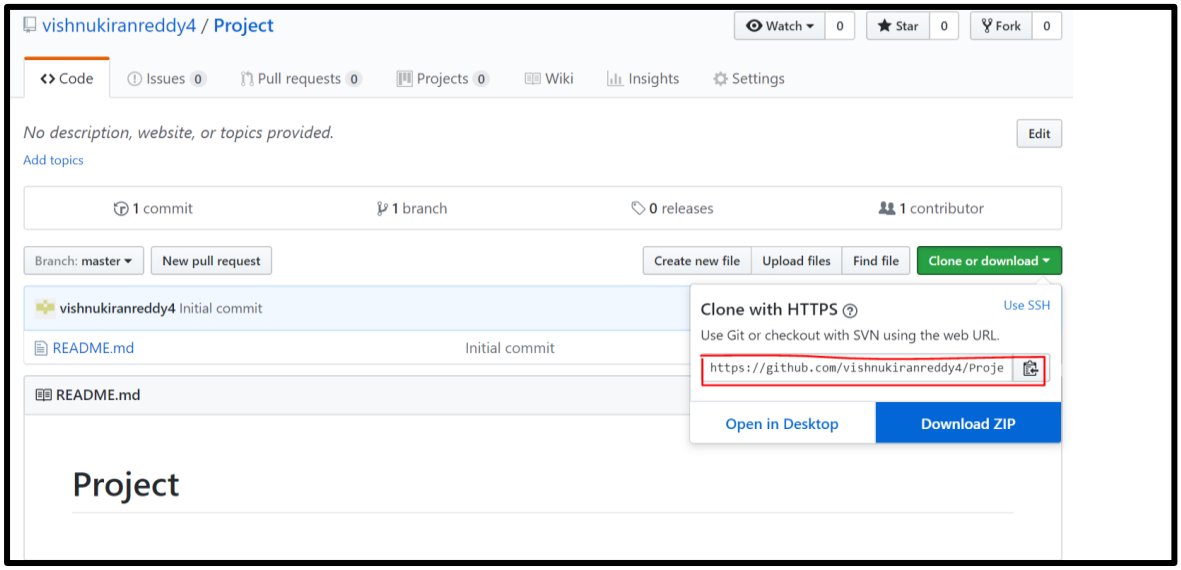
Steps

1. Install Git

1	Following are the steps to install Git: a. Click on Git Download Git from the below URL b. https://git-for-windows.github.io/ c. Start installation d. Select “Git bash” and “Git GUI” e. Select “Use Git from windows command prompt” f. Keep remaining options as default and click Install button. g. If Git installation is completed successfully, then follow these steps. h. Navigate and select Git Bash

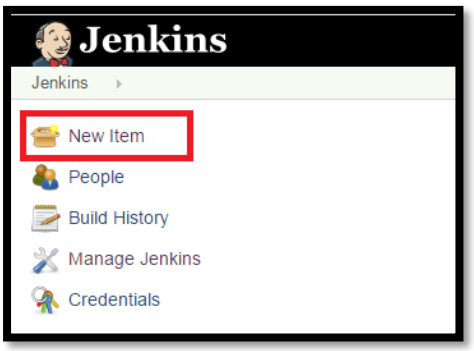
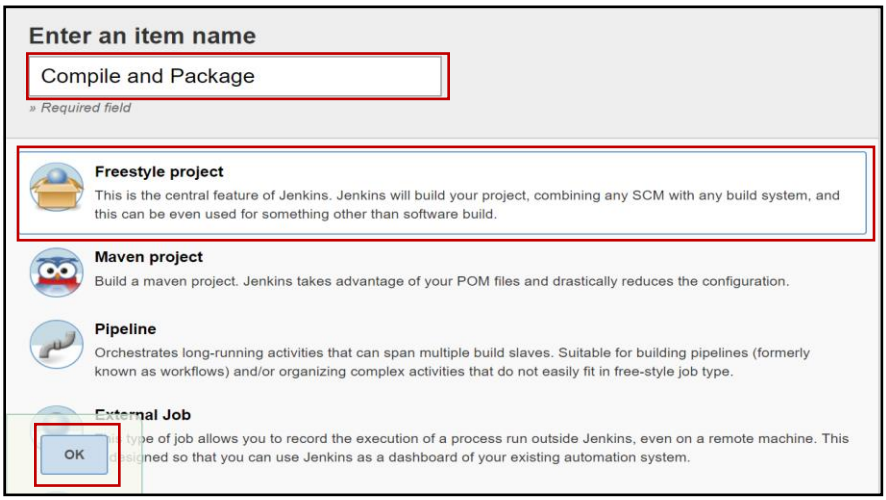
2. Create project in GitHub server

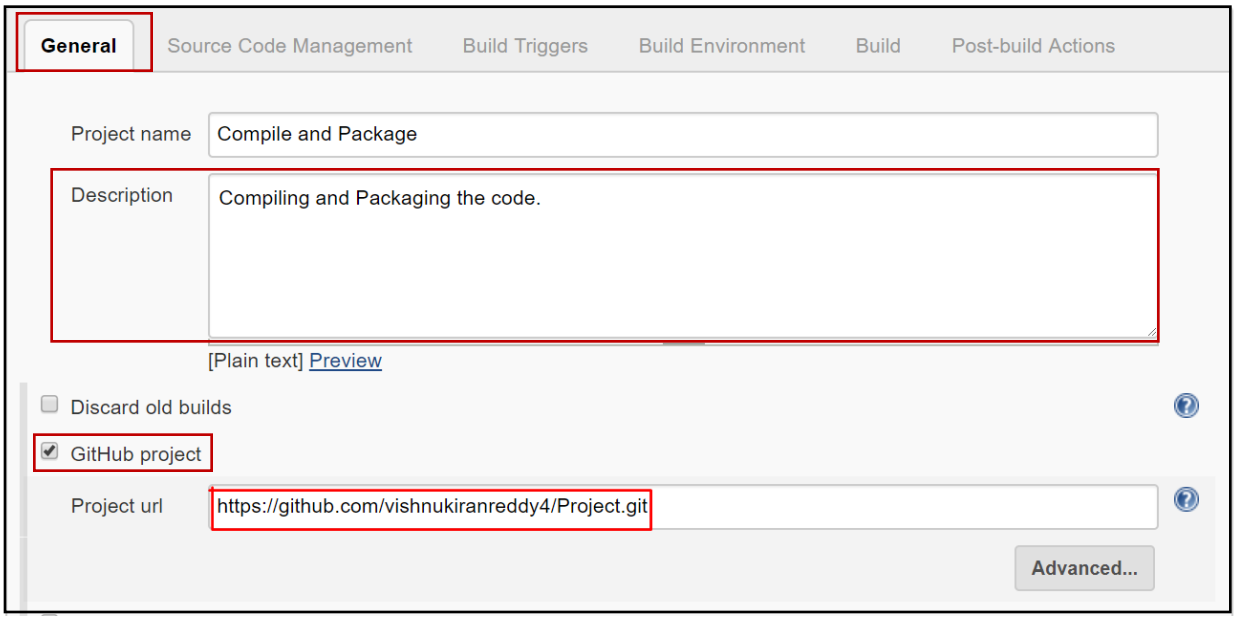
1	Sign in to the GitHub server i. You are required to sign up to GitHub at **https://github.com/** j. While signup, please provide your mail id as an email address. k. (Confirmation link will be sent). l. Once the account is created, login to the account.
2	Create repository a. Click **New repository** from the drop-down list of the GitHub home page  b. Specify the name (example: Project) in the **Repository name** field and click the checkbox next to **Initialize this repository with README** option c. Click the **Create Repository** button to create repository in GitHub server

	
3	Clone the repository Clone the project that you created by using the **git clone** command. The path to the project can be found in GitHub server: The path looks as shown below as per your credentials: git clone https://github.com/vishnukiranreddy4/Project.git 
4	Navigate to Git bash on your workstation and make sure that you have changed the directory to the right directory where you want to clone your project and execute the git clone command a. cd /c/Data b. git clone https://github.com/vishnukiranreddy4/Project.git
5	Change to your Git repository directory: • cd Project
6	Copy the project content into the "Project" folder manually without using Git command
7	After copying content, navigate to git bash and execute the below commands

	• git add . • git status • git config --global user.email <gitloginemail> • git commit -m "Adding project content" • git push origin master Now, the project is pushed to GitHub server Note: Enter password which you have given while creating the account in GitHub.

3. Create job in Jenkins and configure it with the GitHub project

1	Access Jenkins Dashboard using the URL **http://localhost:8082**
2	Select **New Item** from the menu as highlighted in the below image. 
3	a. Type the job name as 'Compile and Package' in the **Enter an item name** field and select **Freestyle Project** as the project type as shown below: b. Click **OK** to create job  This action leads to the **Project Configuration** page, where you can configure settings for build activity
4	To configure job with GitHub project, perform the following tasks: a. In the **Project Configuration page**, under the **General** tab, type the details in the **Description** field b. Check the box next to **GitHub project** c. Provide the path to newly created GitHub project in the **Project url** field as shown below:



4. Configure Source Code Management (SCM) with Git

1

a. In the Project Configuration page, navigate to the **Source Code Management** section
b. Select **Git**
c. Specify the path in **Repository URL**

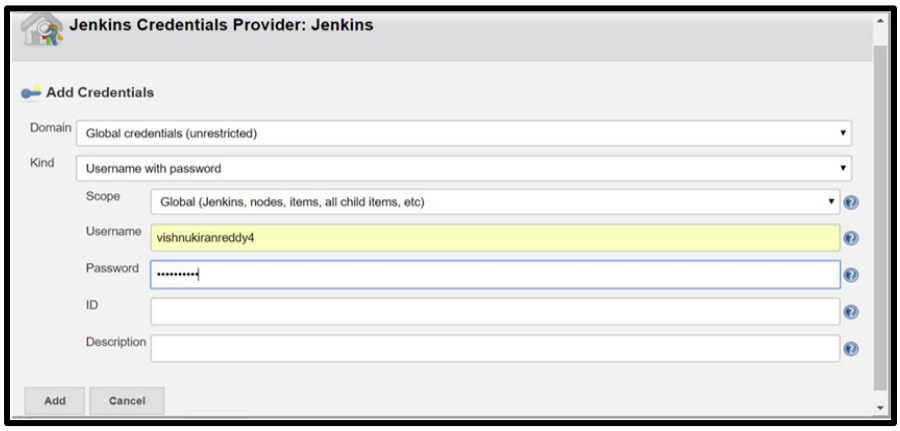
2

Add credentials:

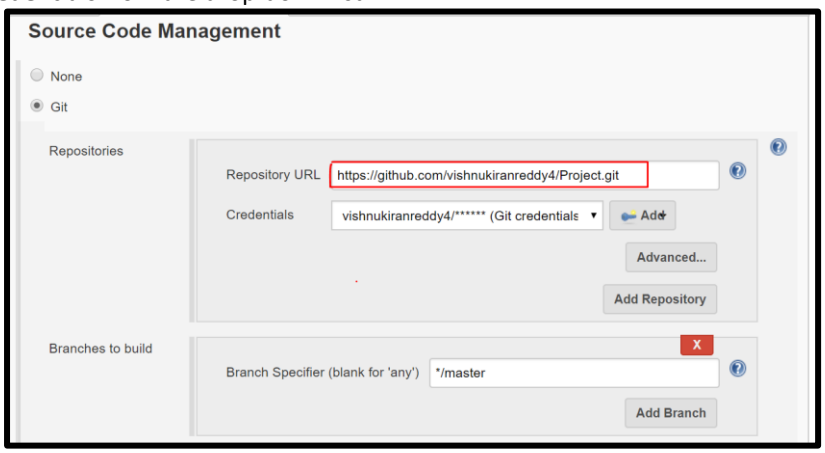
a. Click **Add** button to add username and password of GitHub account
b. Click **Jenkins**

This leads to the **Jenkins Credentials Provider** page.

c. Provide username and password of the GitHub server
d. Click **Add**



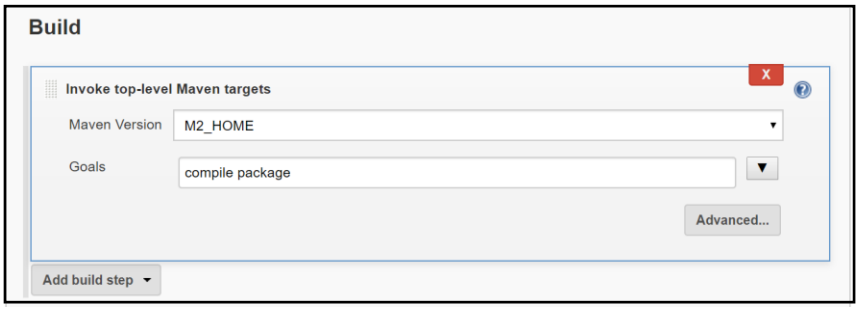
e. Select the **credentials** from the drop-down list



5. Configure build job using Maven

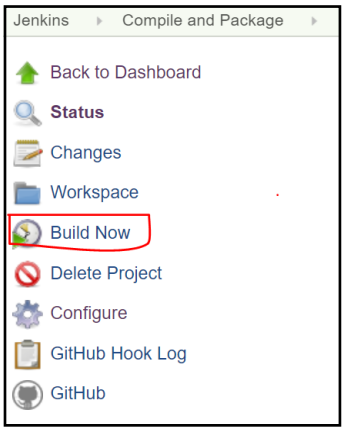
1 Now, build job by using the Maven commands as follows:

a. Navigate to the **Build** section
b. Choose **Invoke top level Maven targets** from the **Add build step** drop-down menu
c. Specify the Maven version as shown in the screen capture below
d. Type the target name as **compile package** against the **Goals** field



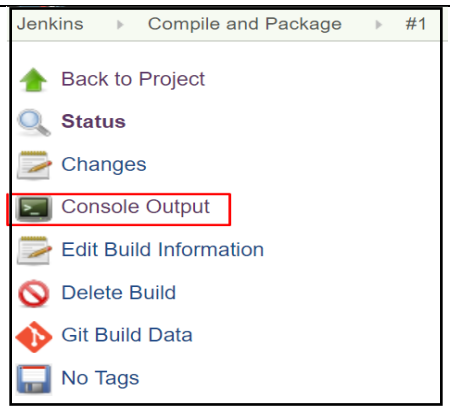
Now, the configuration set up is complete to perform build activity

6. Execute build

1	After the configurations are complete, execute build job manually as follows: a. Click the **Save** button in the Project Configuration page b. Navigate to the Jenkins dashboard and select **Compile and Package** project c. Click **Build Now** to schedule the build to execute immediately  The artifacts (jar/war/ear files) are created on successful build of the project

7. View build results

1	To view build results, click **#1**, the build number under **Build History** of the job in the Jenkins dashboard 
2	Click **Console Output**



This action will retrieve build results and display them in the console:

```
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ HelloWorld ---
[INFO] Packaging webapp
[INFO] Assembling webapp [HelloWorld] in [C:\Jenkins\workspace\Compile and Package\target\HelloWorld]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Jenkins\workspace\Compile and Package\src\main\webapp]
[INFO] Webapp assembled in [289 msecs]
[INFO] Building war: C:\Jenkins\workspace\Compile and Package\target\HelloWorld.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.847 s
[INFO] Finished at: 2017-10-13T18:24:43+05:30
[INFO] Final Memory: 16M/154M
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

End of Module 3

Module 4

Exercise 4.1: Create a Job and Analyze Code in Jenkins

Perform activities to analyze code using SonarQube Scanner in Jenkins

Prerequisite:

1. SonarQube and Sonar Scanner to be stored in the C:\Softwares of the local system
2. The project in GitHub server used in Exercise 3.1 is considered in this exercise to perform code analysis

Note:

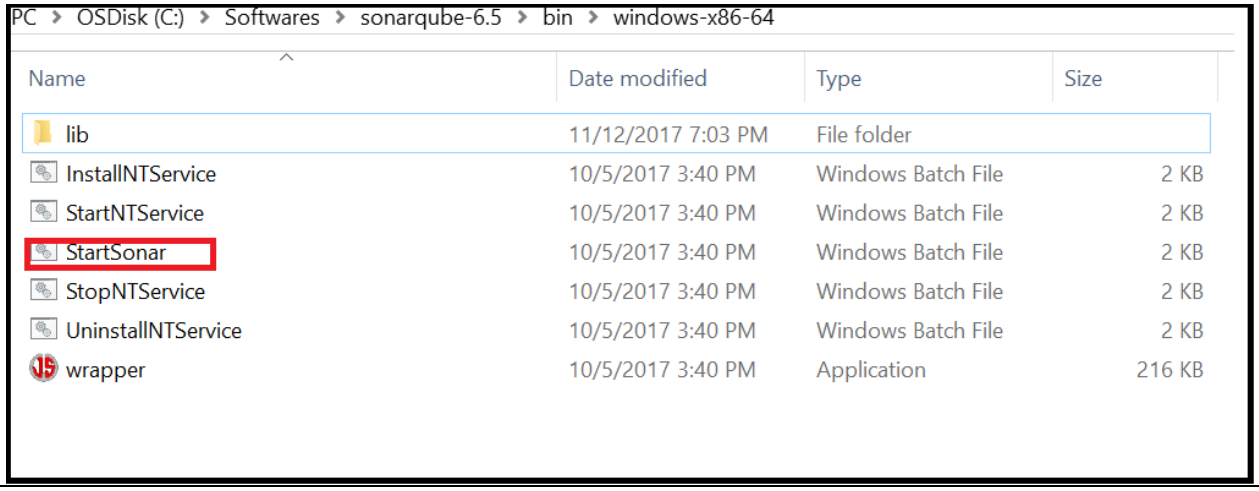
1. SonarQube and SonarQube Scanner will be provided by faculty during session
2. The SonarQube 6.5 version is used in this exercise

Walkthrough

1. Install SonarQube
2. Configure Jenkins with SonarQube
3. Configure Jenkins with SonarQube Scanner
4. Install SonarQube Scanner plugin
5. Create job in Jenkins and configure it with the GitHub project
6. Configure SCM with Git
7. Perform code analysis using SonarQube Scanner
8. View analysis results

Steps

1. Install SonarQube

1	Navigate to **C:\Softwares\sonarqube-6.5\bin\windows-x86-64** folder 
2	a. Click on **StartSonar** to start the SonarQube server. b. Once we click on StartSonar, SonarQube server will be up and running.

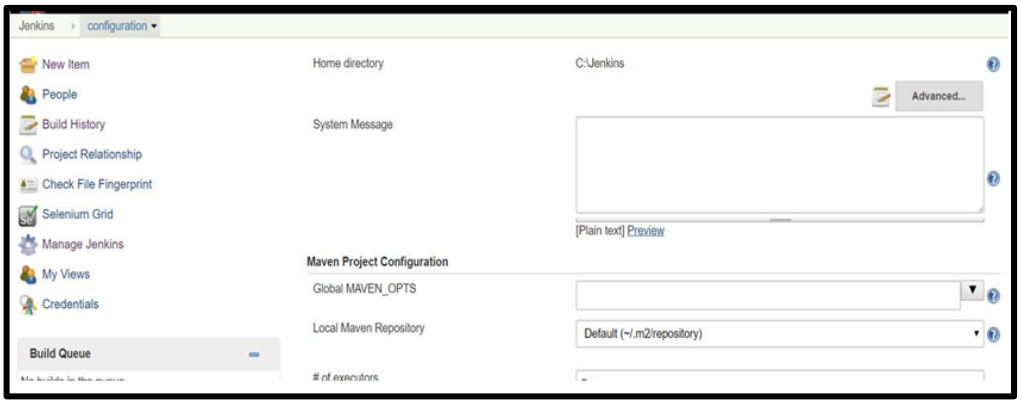
```
jvm 1    | 2017.11.12 18:51:28 INFO  app[][o.s.a.SchedulerImpl] Process[web] is up
jvm 1    | 2017.11.12 18:51:28 INFO  app[][o.s.a.p.JavaProcessLauncherImpl] Launch pr
e1.8.0_151\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Xmx512m -Xms128m
ava.io.tmpdir=C:\Java\sonarqube-6.5\temp -cp ./lib/common/*;./lib/server/*;./lib/ce/*
h2-1.3.176.jar org.sonar.ce.app.CeServer C:\Java\sonarqube-6.5\temp\sq-process7585436
jvm 1    | 2017.11.12 18:51:35 INFO  app[][o.s.a.SchedulerImpl] Process[ce] is up
jvm 1    | 2017.11.12 18:51:35 INFO  app[][o.s.a.SchedulerImpl] SonarQube is up
```

c. You can access SonarQube in the path: http://localhost:9000

d. **Log in** to the SonarQube server with the following credentials:
Username: admin
Password: admin

2. Configure Jenkins with SonarQube

1	Open the Jenkins dashboard in the URL http://localhost:8082
2	Navigate to **Manage Jenkins → Configure System**  This results in the **Configuration** page, where you can choose the components to configure Jenkins with.

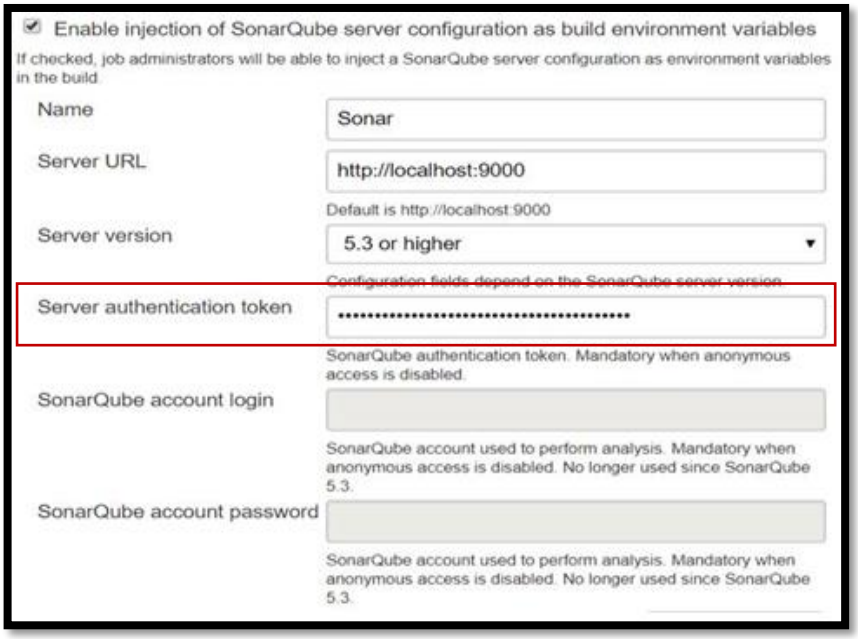


3

a. In the Configuration page, navigate to the **SonarQube Servers** section

b. Click the box next **to Enable injection of SonarQube server configuration as build environment variables**

c. Click **Add SonarQube** and provide required details as shown below**:**



Note: **Server authentication token** is generated in SonarQube server. To generate token, perform the following steps:

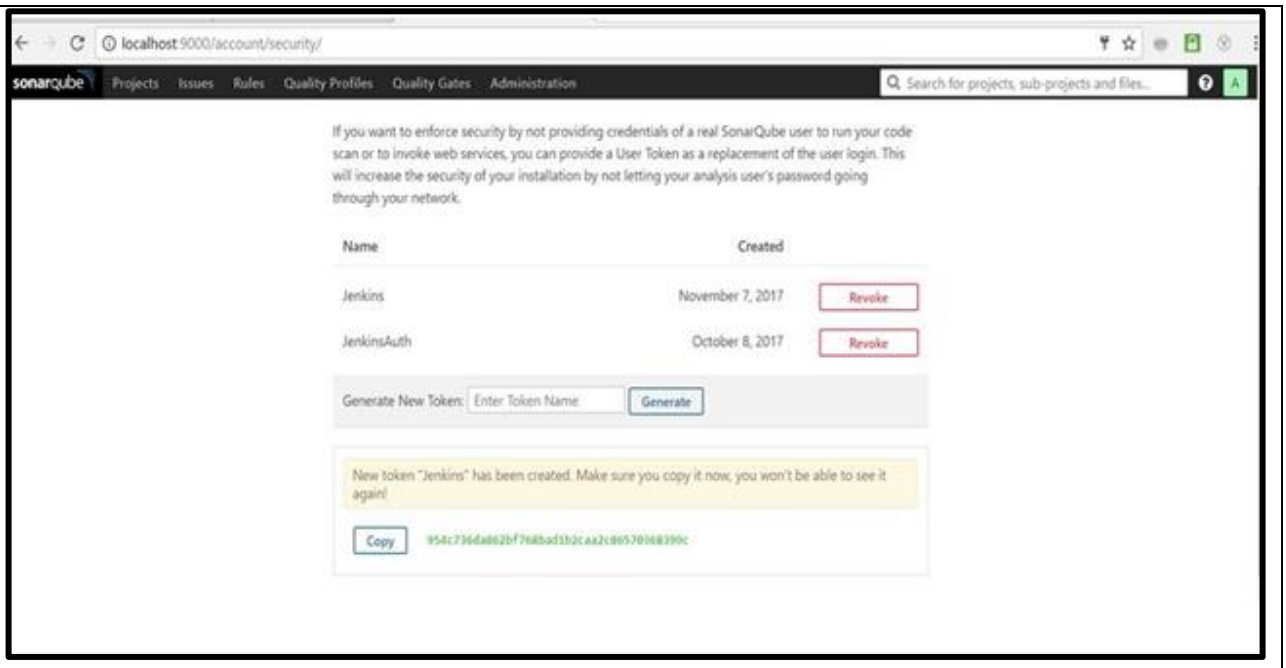
a. Access SonarQube server by using the link http://localhost:9000

b. **Log in** to the SonarQube server with the following credentials: username: admin, password: admin

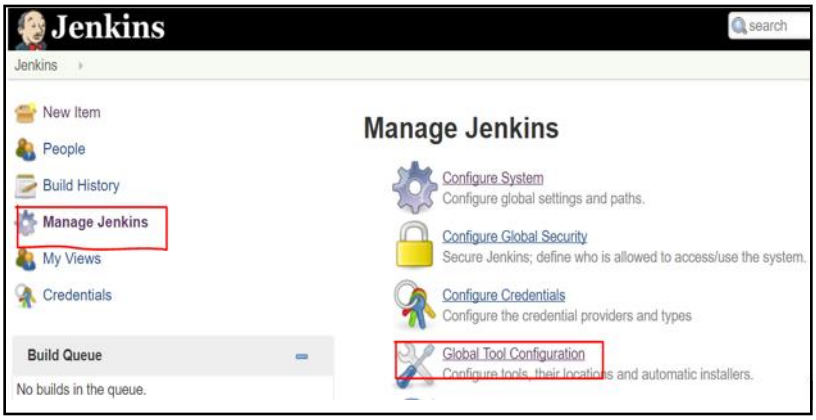
c. Click **A** at the top right-hand corner

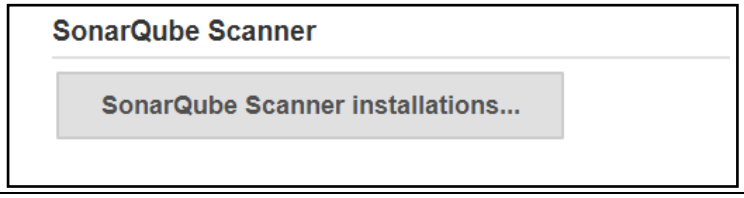
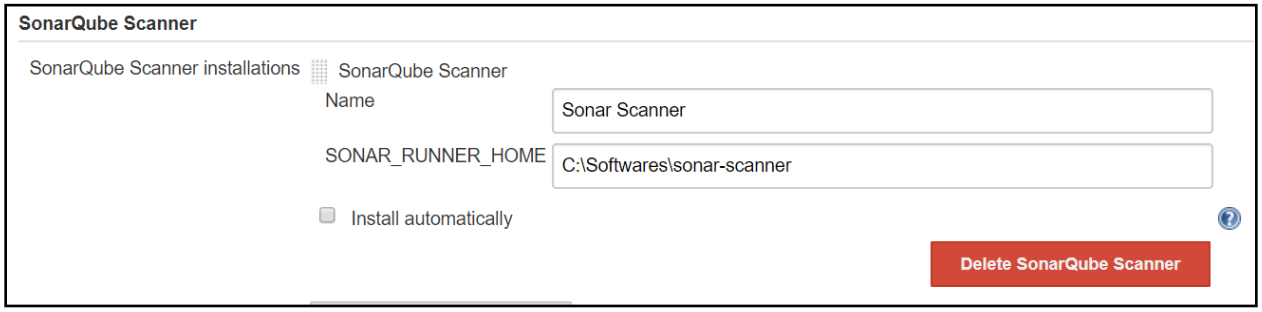
d. Navigate to **My Account→Security**

e. Specify the token name and then click the **Generate** button to generate the token

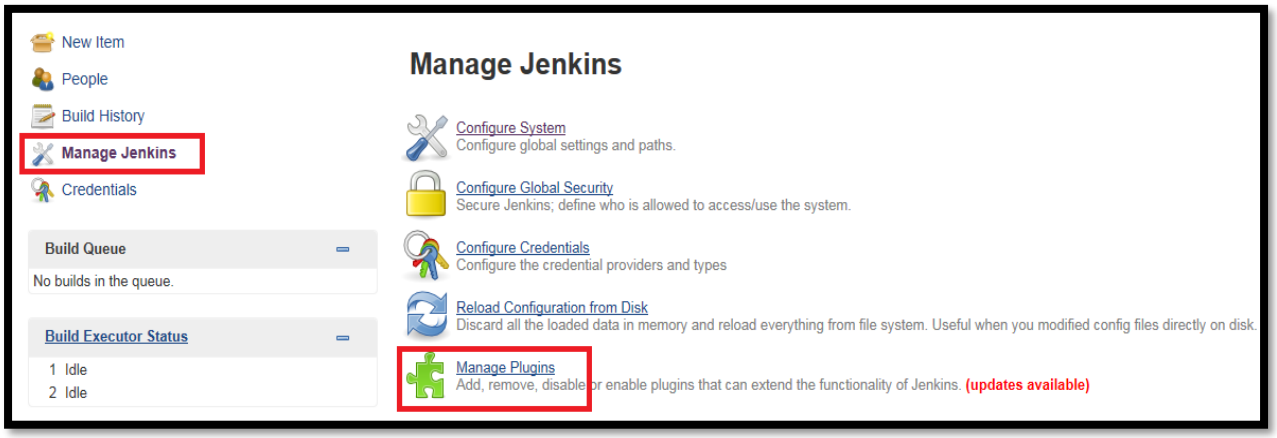
	 Copy the token and use it while configuring SonarQube server in Jenkins
5	Click the **Save** button to save the SonarQube server configuration

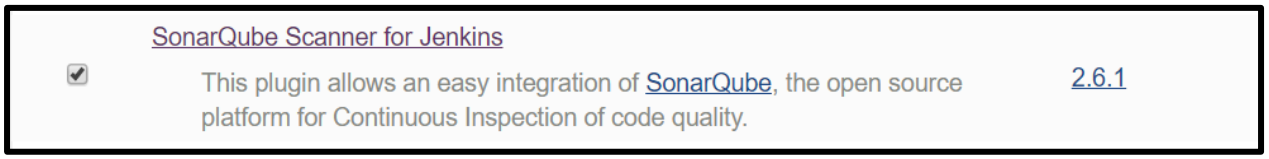
3. Configure Jenkins with SonarQube Scanner

1	Open the Jenkins dashboard in the URL http://localhost:8082
2	Navigate to **Manage Jenkins → Global Tool Configuration** option  This results in the **Global Tool Configuration** page, where you can choose the components to configure Jenkins with.
3	In the Global Tool Configuration page, navigate to the **SonarQube Scanner** section and Click **SonarQube Scanner Installations**

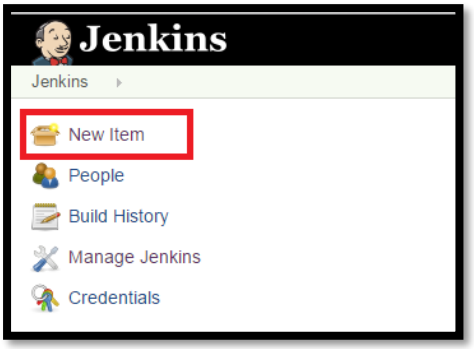
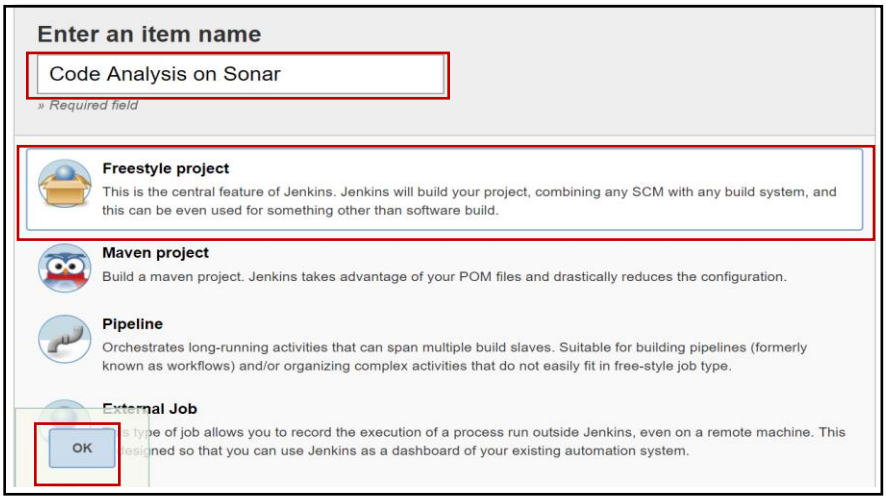
	
4	a. Uncheck the box next to **Install automatically** b. Specify the name and path of the SonarQube Scanner in the **SONAR_RUNNER_HOME** field. The example is shown in the following capture: 
5	Click the **Save** button to save the SonarQube Scanner configuration

4. Install SonarQube Scanner plugin

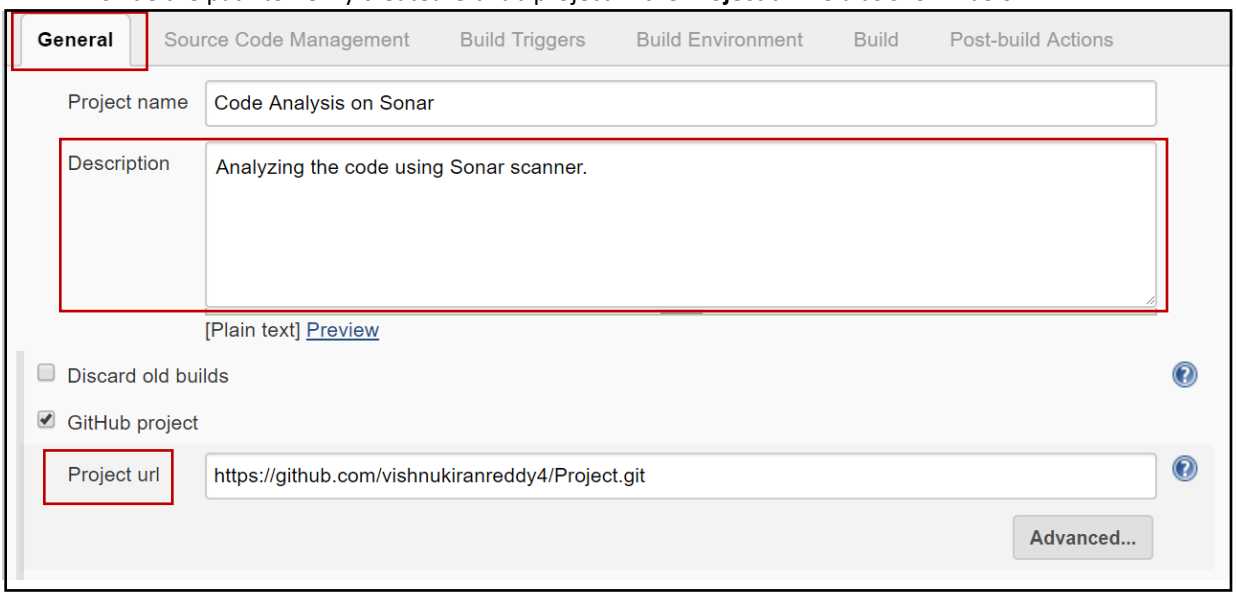
1	In Jenkins dashboard, navigate to **Manage Jenkins → Manage Plugins** option. 
2	a. Click the **Available** tab under Plugin Manager page and then search and select **SonarQube Scanner for Jenkins** plugin

	 b. Click **Install without restart** to install the plugin

5. Create job in Jenkins and configure it with the GitHub project

1	Access Jenkins Dashboard using the URL **http://localhost:8082**
2	Select **New Item** from the menu as highlighted in the below image. 
3	c. Type the job name as 'Compile and Package' in the **Enter an item name** field and select **Freestyle Project** as the project type as shown below: d. Click **OK** to create job  This action leads to the Project Configuration page, where you can configure settings for build activity
4	To configure job with GitHub project, perform the following tasks: d. In the **Project Configuration page**, under the **General** tab, type the details in the **Description** field e. Check the box next to **GitHub project**

f. Provide the path to newly created GitHub project in the **Project url** field as shown below:



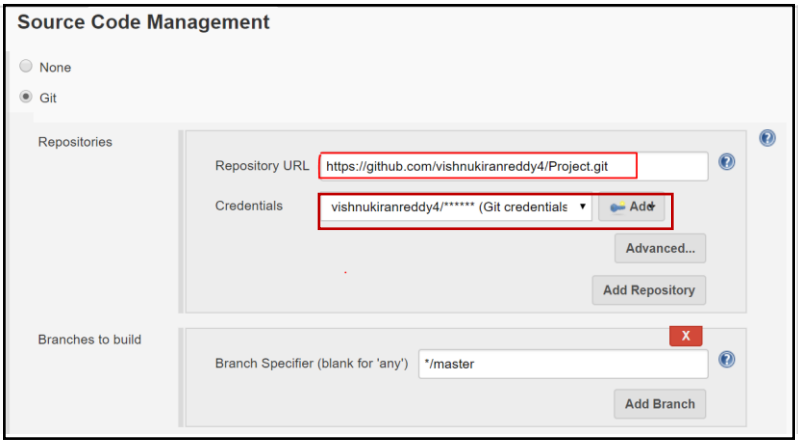
6. Configure SCM with Git

1

a. In the Project Configuration page, navigate to the **Source Code Management** section
b. Select **Git**

2

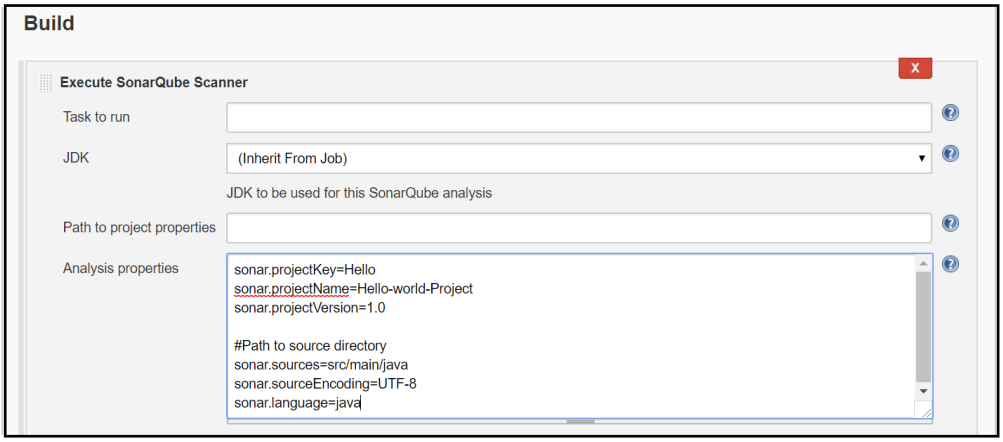
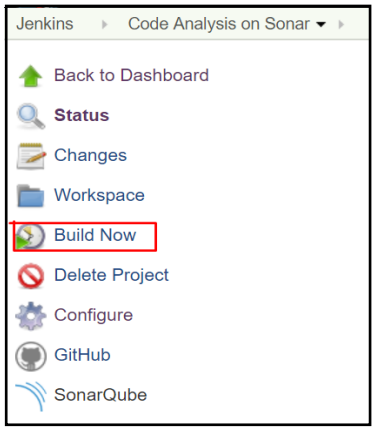
a. Specify the path in **Repository URL**
b. To add credentials, click **Add** button to add username and password of GitHub account
c. Select the credentials from the drop-down list



7. Perform code analysis

1

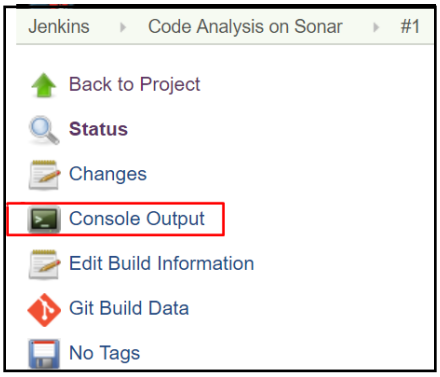
To perform code analysis by using SonarQube Scanner, follow the steps below:
a. Navigate to the **Build** section

	b. Choose **Execute SonarQube Scanner** from the **Add build step** drop-down menu c. Provide the details for **Analysis Properties** as shown below  Now, the configuration set up is complete to perform code analysis. d. Click the **Save** button in the Project Configuration page
2	After the configurations are complete, execute analysis manually as follows: a. Navigate to the Jenkins dashboard and select **Code Analysis on Sonar** project b. Click **Build Now** to execute code analysis 

8. View analysis results

1	To view build results, click **#1**, the build number under **Build History** in the Jenkins dashboard for the project, Code Analysis on Sonar

2 Click **Console Output**



This action will retrieve results in console. Access the below link in the console output to see the analysis results

```
INFO: Analysis report uploaded in 54ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard/index/Hello
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis
report
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AV8V8K4hbPHDgewYO0Bz
INFO: Task total time: 6.043 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 9.260s
INFO: Final Memory: 54M/265M
INFO: ------------------------------------------------------------------------
Finished: SUCCESS
```

End of Module 4

Module 5

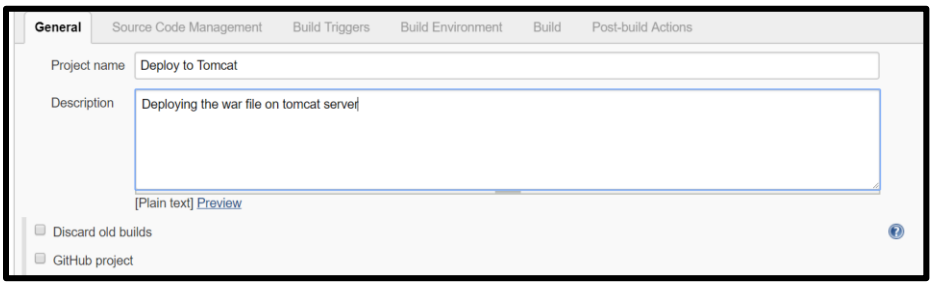
Exercise 5.1: Create a Job to Deploy WAR File in Jenkins

Perform activities to create a project and deploy it in Tomcat server

Prerequisite:

1. Add the **Deploy to Container Plugin** and **Copy Artifacts Plugin** to Jenkins to perform deployment (Install the plugin as mentioned in the steps in the exercise 2.3)
2. Tomcat should be installed in your local system
3. The artifact created in Exercise 3.1 is considered in this exercise for deployment in Tomcat
4. A 'Free Style Project' named **Deploy to Tomcat** should be created in Jenkins for deployment (Please follow the steps in exercise 3.1 to create the job)

Note: Tomcat will be shared by the faculty during the session



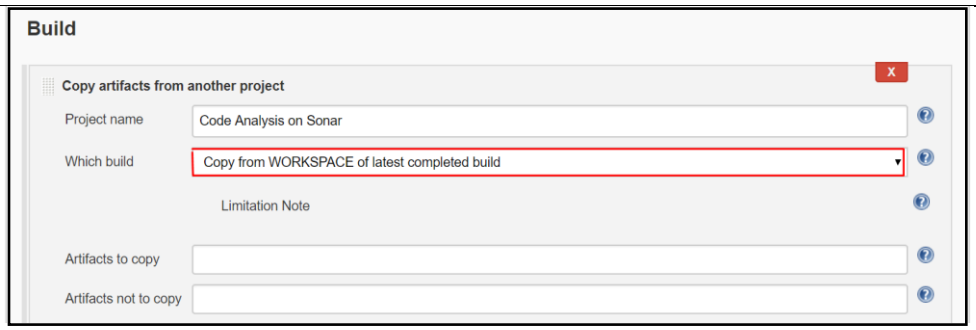
Walkthrough

1. Perform build activities to deploy the project
2. Perform post build tasks to deploy the project
3. Execute project deployment
4. View results

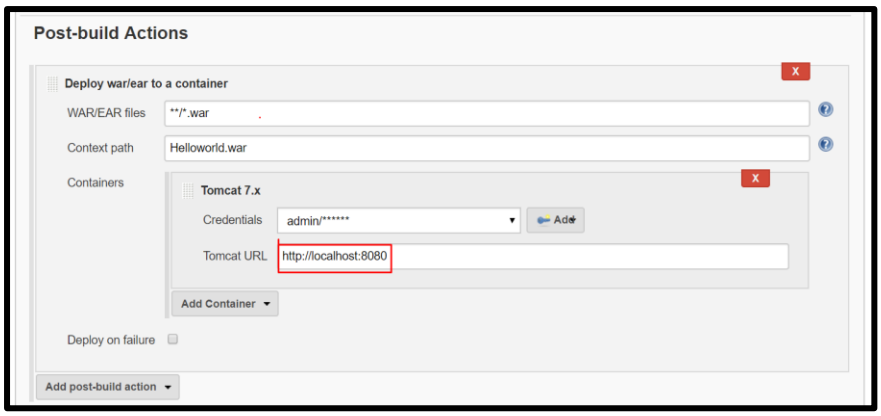
Steps

1. Perform build activities to deploy the project

1	In the Project Configuration page, navigate to the **Build** section
2	Choose **Copy artifacts from another project** from the **Add build step** drop-down list
3	a. Specify **Code Analysis on Sonar** as **Project Name** (i.e., Project from which we should copy the artifacts) b. Select **Copy from WORKSPACE of the latest completed build** from the drop-down list

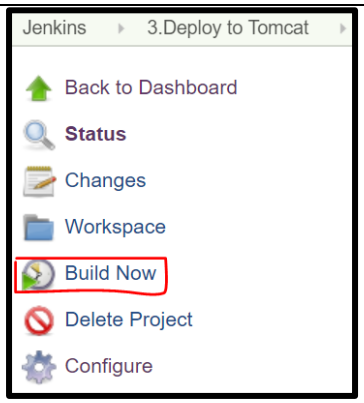
	

2. Perform post build tasks to deploy the project

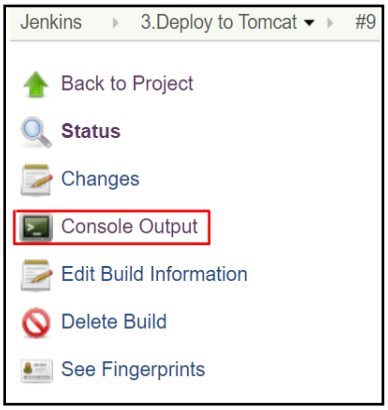
1	In the Project Configuration page, navigate to the **Post-build Actions** section
2	Configure Tomcat to deploy project a. From the **Add Post-build action** drop-down list, choose **Deploy war/ear to a container** b. Specify **WAR/EAR files** as ****/*.war** c. Specify **Context Path** as Helloworld.war d. Click **Add Container** drop-down menu and choose **Tomcat 7.x** and provide the details as shown below e. Click **Add** to add credentials of Tomcat and select it from the drop-down list  f. Click **Save** to save the configurations

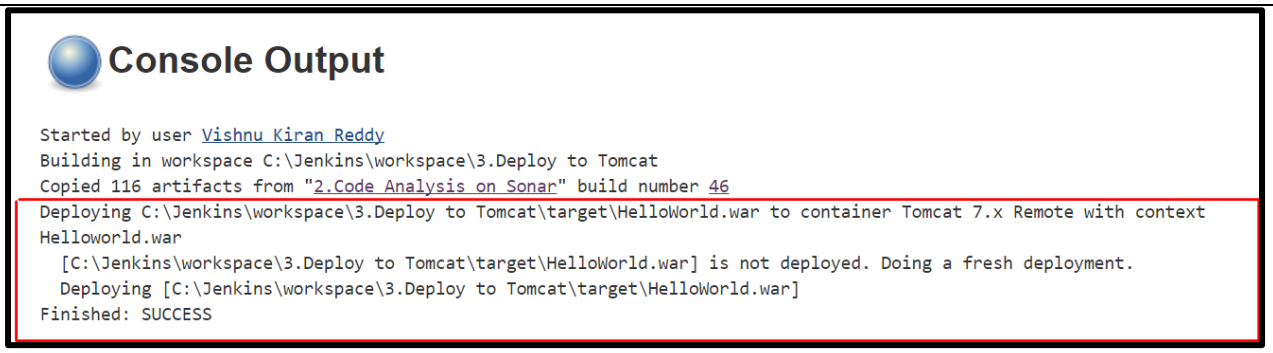
3. Execute project deployment

1	Now that the deployment configurations are complete, execute deployment job manually by following the below steps: a. In the Jenkins dashboard, navigate to the **Deploy to Tomcat** project b. Schedule the deployment to be executed immediately by clicking **Build Now**

4. View results

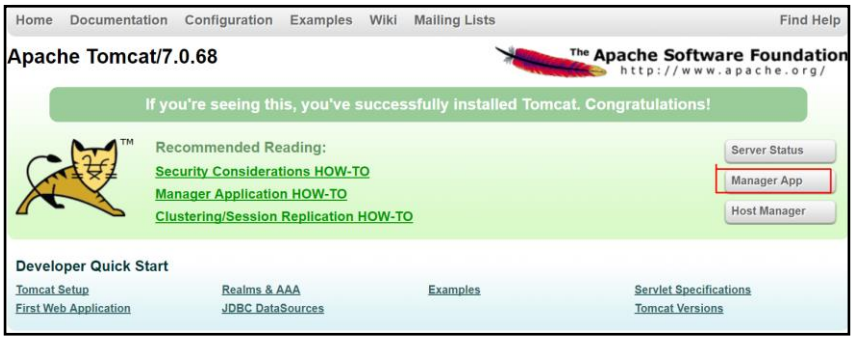
1	To view build results, click **#1**, the build number under **Build History** in the Jenkins dashboard for the project, Deploy to Tomcat 
2	Click **Console Output**  This action will retrieve results in console. Access the below link in the console output to see the analysis results



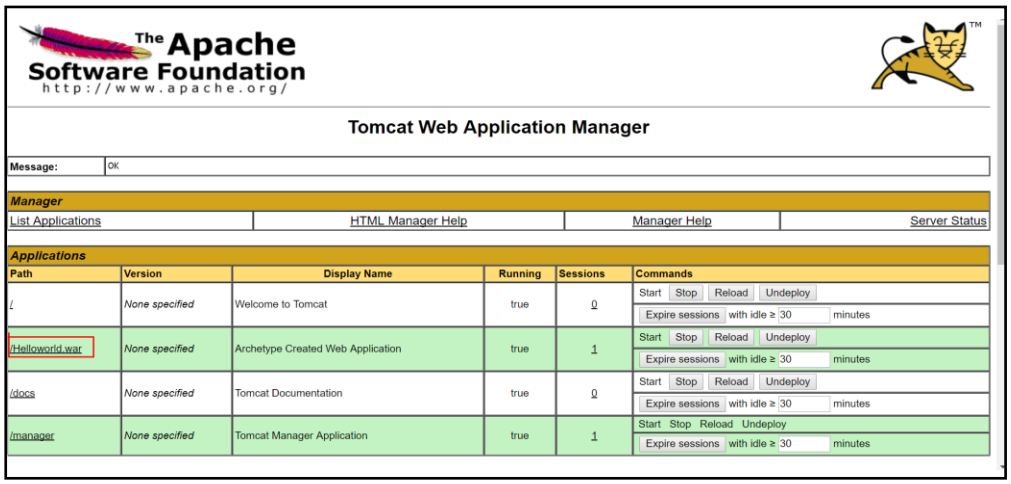
3 To verify deployment, perform the following tasks:

a. Go to browser and access the link http://localhost:8080 to see the Tomcat home page

b. Click the **Manager App** icon to see the deployed war file (i.e., Helloworld.war)



c. Click the **Helloworld.war** file to check the output



You can see that the deployed application is up and running

End of Module 5

Module 6

Exercise 6.1: Test application with Selenium

Scenario

Perform activities to test with Selenium and view test results using TestNG in Jenkins

Prerequisite:

- The project in GitHub server used in Exercise 3.1 is considered in this exercise to test the application with selenium.
- Include the plugin, TestNG Results Plugin to view TestNG results
- Add **Selenium Plugin** to test the application with Selenium
- **Mozilla Firefox** should be installed for testing

Note:

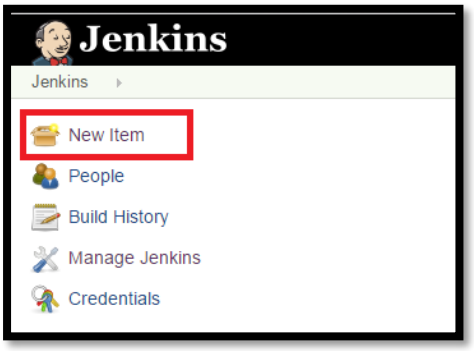
1. The test results can be viewed in:
   a. Console Output
   b. TestNG Results
2. In this exercise, TestNG is used to view test results.

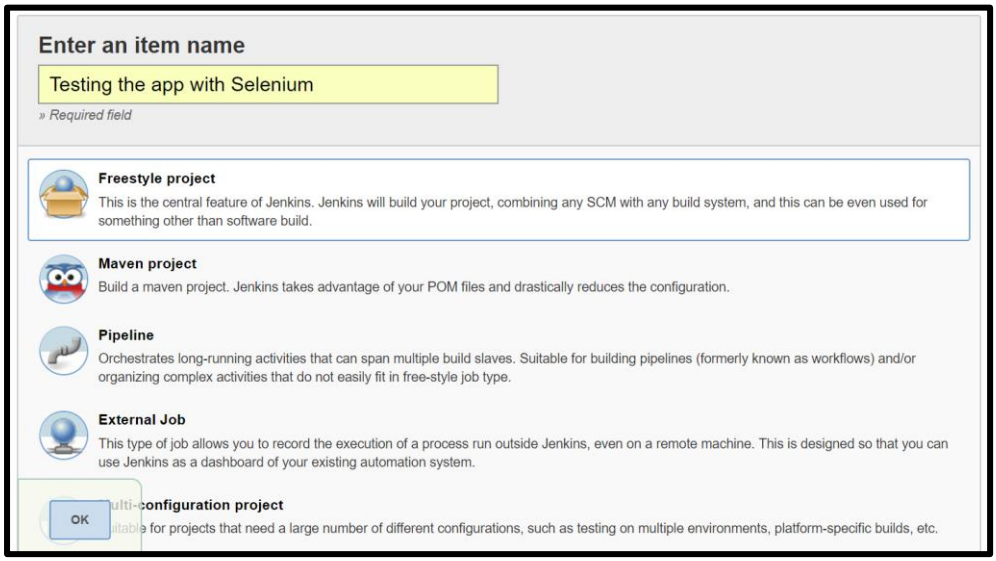
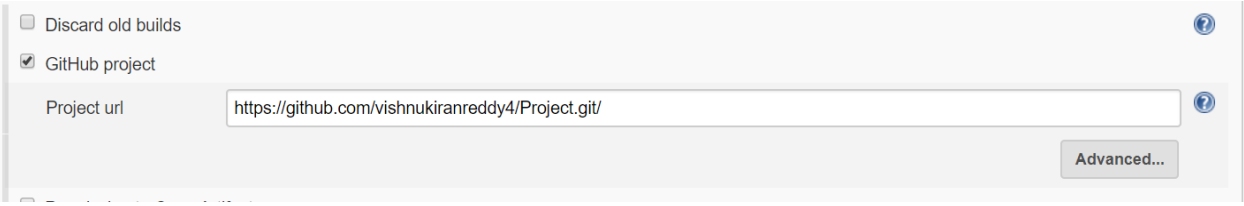
Walkthrough

1. Create job in Jenkins and configure it with the GitHub project
2. Configure Source Code Management with Git
3. Configure build with Maven and test results with TestNG
4. View results using TestNG

Steps

1. Create job in Jenkins and configure it with the GitHub project

2	Select **New Item** in the Jenkins dashboard from the menu as highlighted in the below image. 

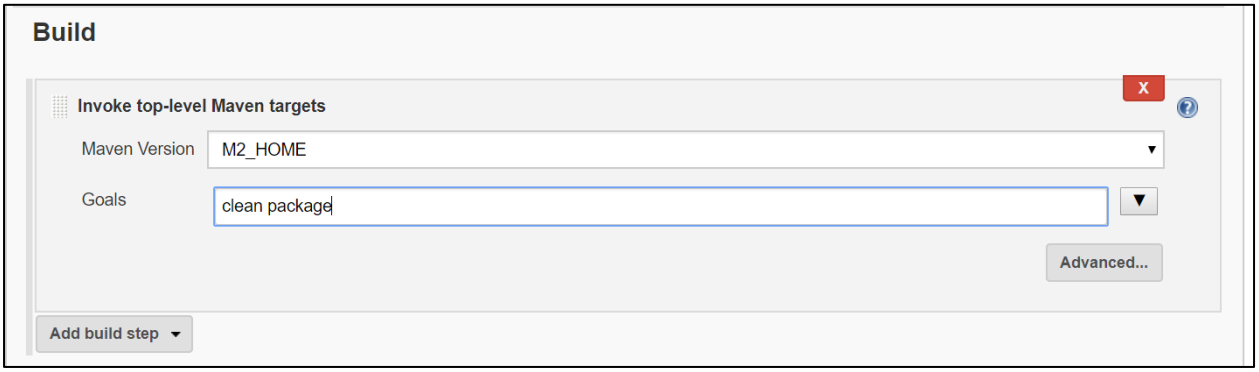
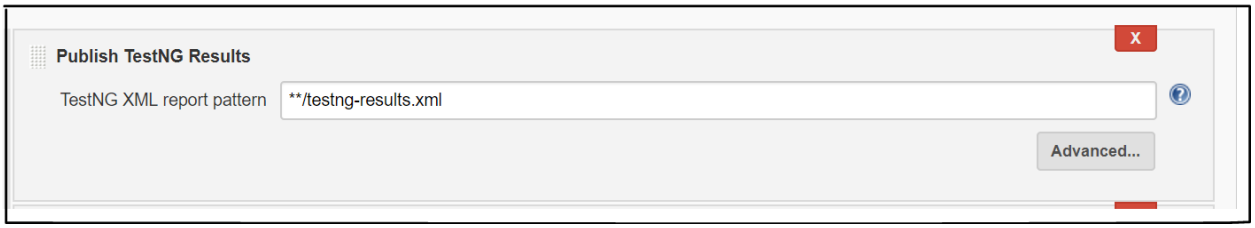
3	a. Type the job name as **Testing the app with Selenium** in the **Enter an item name** field and select **Freestyle Project** as the project type as shown below: b. Click **OK** to create job  This action leads to the Project Configuration page, where you can configure settings for testing activity
4	To configure job with GitHub project, perform the following tasks: a. In the **Project Configuration** page, under the **General** tab, type the details in the **Description** field b. Check the box next to **GitHub project** c. Provide the path to newly created GitHub project/repository in the **Project url** field as shown below: 

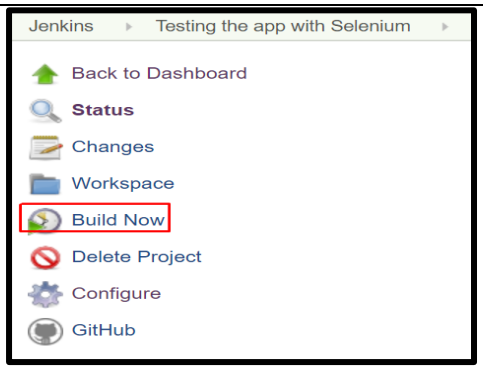
2. Configure Source Code Management with Git

4	To configure SCM with GitHub project, perform the following tasks:
	a. In the **Project Configuration page**, under the **Source Code Management** section:
	  a. Select **Git**
	  b. Specify Repository URL
	  c. Select the credentials from dropdown
	  d. Click **Save** to save the configuration

3. Configure build with Maven and test results with TestNG

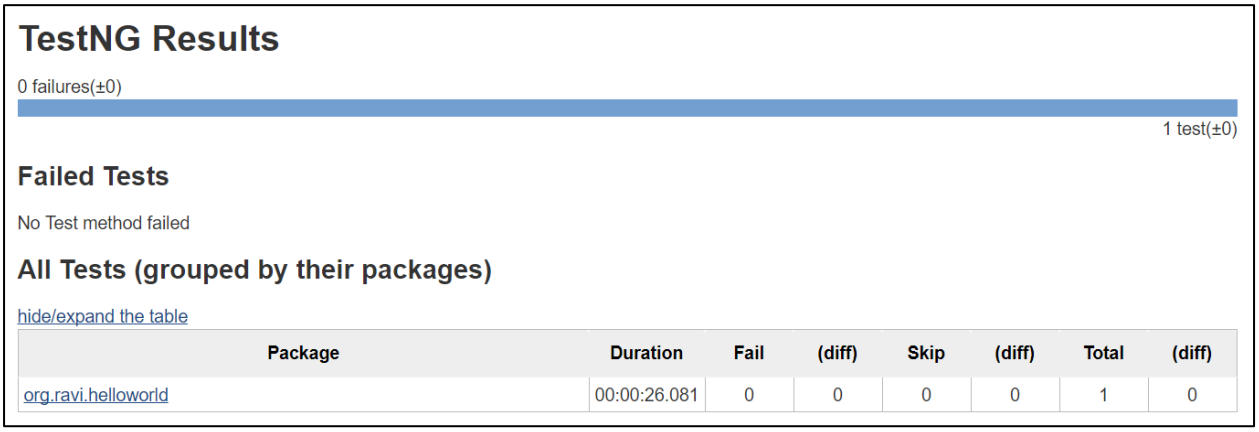
1	To build job using maven commands, do the following steps. a. Navigate to **Build** section b. Choose **Invoke top-level Maven targets** from **Add build step** drop-down list. c. Specify the Maven version and type target name as shown below to execute clean and package goals in Maven 
2	d. In the Project Configuration page, navigate to the Post-build Actions section e. From the Add post-build action drop-down items, select Publish TestNG Results f. Specify the XML report pattern as shown, to save results during build execution 
4	Once configurations are completed, execute build job manually by following the below steps: a. Click **Save** b. Schedule the build to be executed immediately by clicking the **Build Now** menu item



5 View the generated test results by following the below steps:

a. Select the execute build number
b. Click **TestNG Results** link

The test TestNG results are as shown below:



TestNG Results

0 failures(±0)

1 test(±0)

Failed Tests

No Test method failed

All Tests (grouped by their packages)

hide/expand the table

Package	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
org.ravi.helloworld	00:00:26.081	0	0	0	0	1	0

End of Module 6

Module 7

Exercise 7.1: Create Job Pipeline

Scenario

Perform activities to create pipeline and configure jobs to execute in the Pipeline.

Prerequisite:

- Include the plugin**, Build Pipeline Plugin** in Jenkins configuration to create and work with Pipeline
- The jobs to include in the Pipeline should be created before creating Pipeline

Note: This exercise considers 3 jobs, Compile and Package, Code Analysis on Sonar and Deploy to Tomcat to execute in the Pipeline

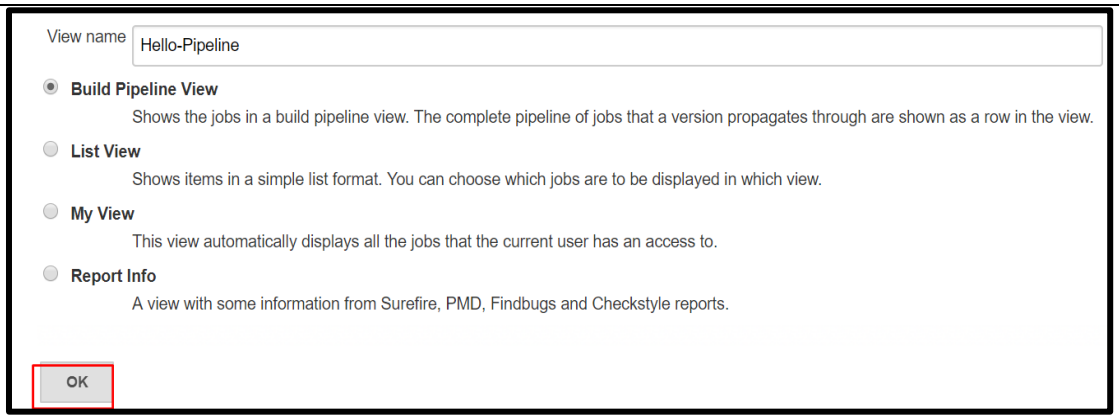
Walkthrough

1. Create Pipeline
2. Configure sequence of jobs in Pipeline
3. Execute Pipeline – 1 Touch and 0 Touch deployment

Steps

1. Create Pipeline

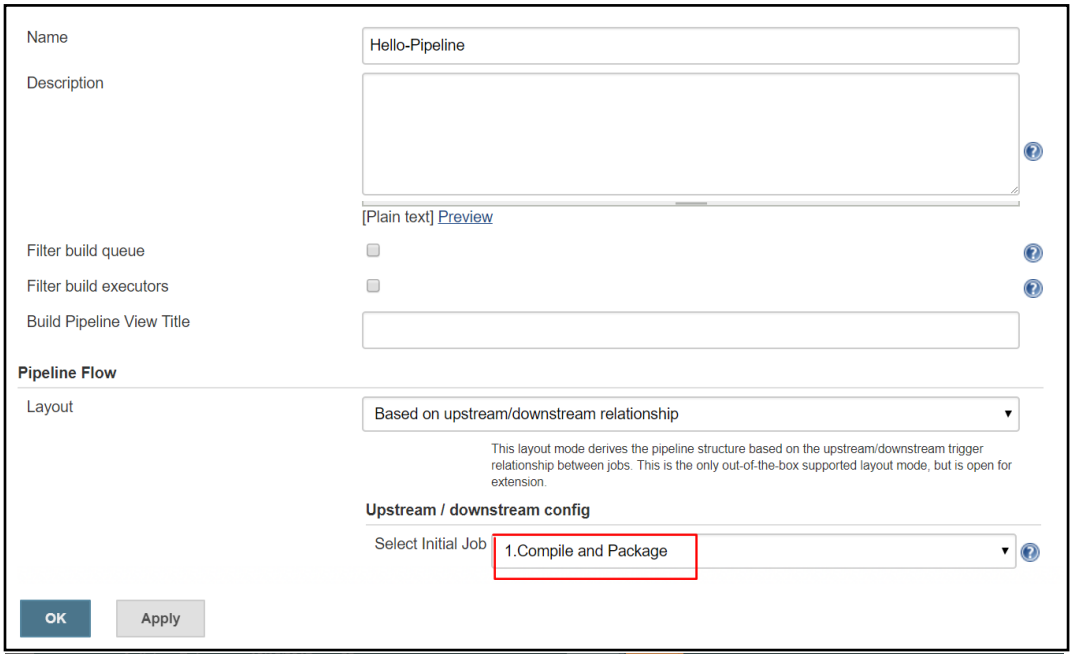
1	To create a pipeline, click on the "+" option in the Jenkins dashboard 
	a. To name the Pipeline, provide the details for **View name** as Hello-Pipeline **b.** Check the box next to **Build Pipeline View** **c.** Click **OK**



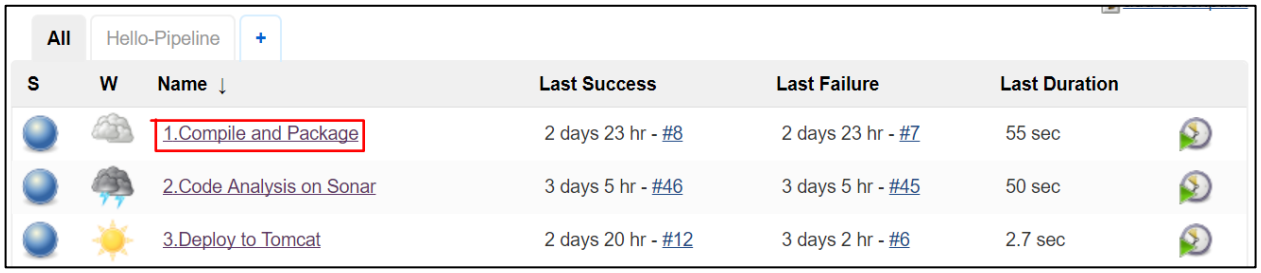
This will successfully create a pipeline and Jenkins will display configurations screen immediately

2. Configure sequence of jobs in Pipeline

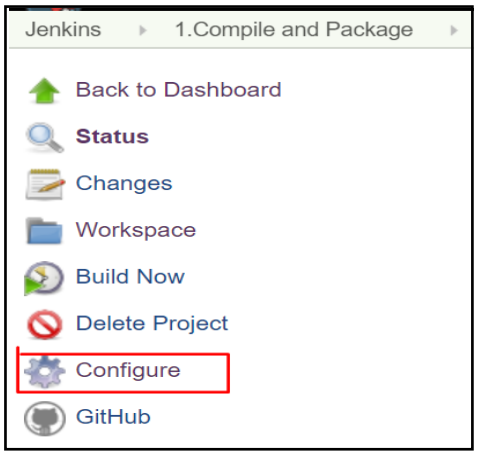
1. Select the first job to execute in the Pipeline.
   a. In this exercise, select **compile and Package** for the **Select Initial Job** field to indicate it as the first job to execute in the Pipeline, Hello-Pipeline as shown below:
   b. Click **OK**



2. After selecting the initial job, you configure the first job
   a. Navigate to the Jenkins Dashboard
   b. Click the first job, **Compile and Package** as shown below:

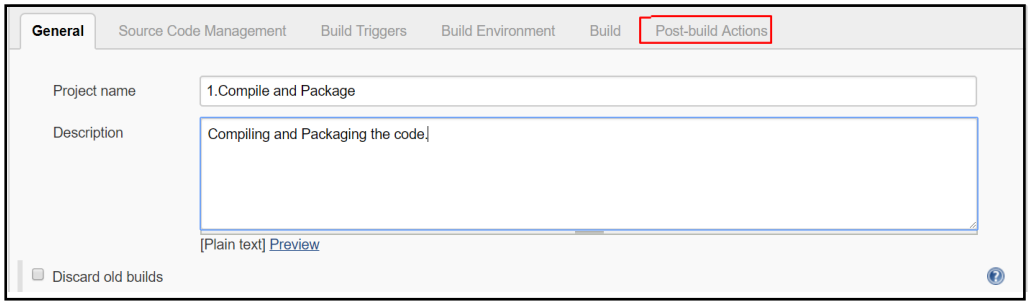


c. Click **Configure**



3 Next, to configure the second job, **Code Analysis on Sonar** in the Pipeline,

a. Navigate to the **Post-build Actions** section in the Project Configuration page of the Compile and Package job

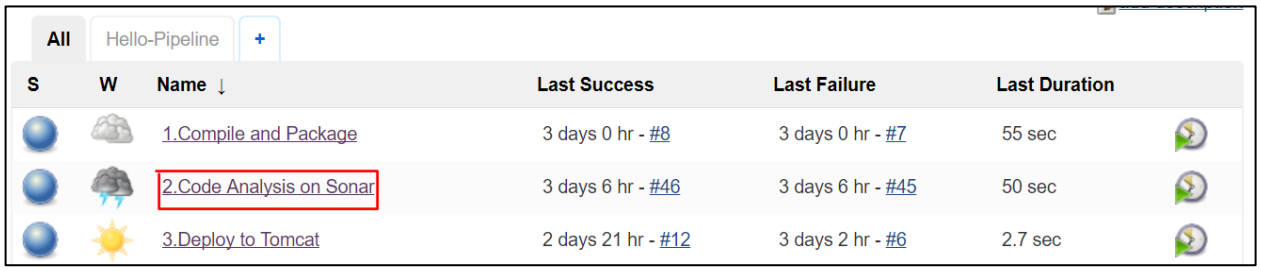


b. From the **Add post-build action** dropdown list, choose **Build other projects**

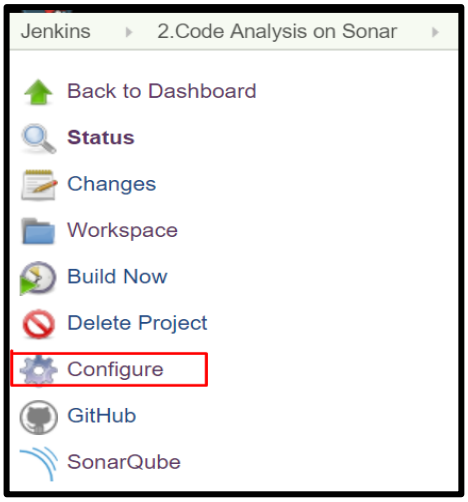
c. Specify the next job which we want to build after the successful execution of first job, Code Analysis on Sonar as shown below

d. Click **Save**

e. Navigate to the Jenkins dashboard and click the job, **Code Analysis on Sonar**



f. Click **Configure**

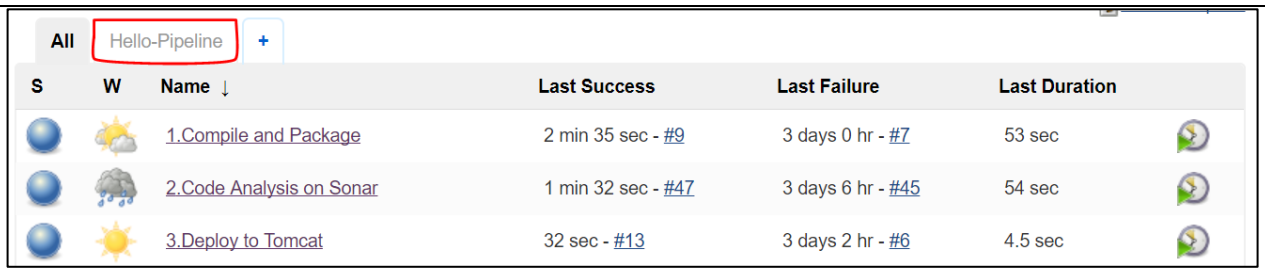


This action leads to the Configuration Page of the Code Analysis on Sonar job, from where you can configure the next job in Pipeline

g. Similarly, by following the same steps mentioned above, configure the third job, Deploy to Tomcat on the Project Configuration page of the Code Analysis on Sonar job

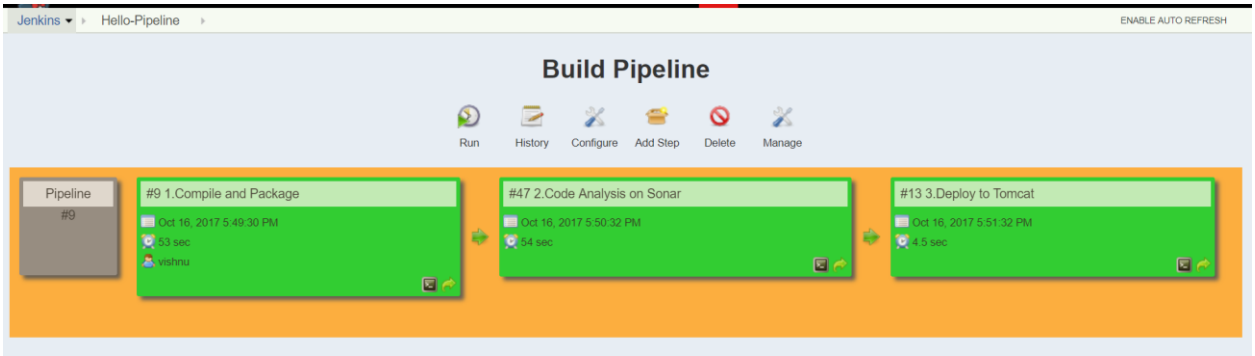
3. Execute Pipeline – 1 Touch and 0 Touch deployment

1	After all the jobs are configured, you can execute jobs as per the specified configuration: a. Navigate to the Jenkins dashboard b. Click the **Hello-Pipeline** created c. Click the **Run** icon (This is for **1 touch** deployment)



d. Click **Run**

This action triggers jobs to execute in the specified sequential order.

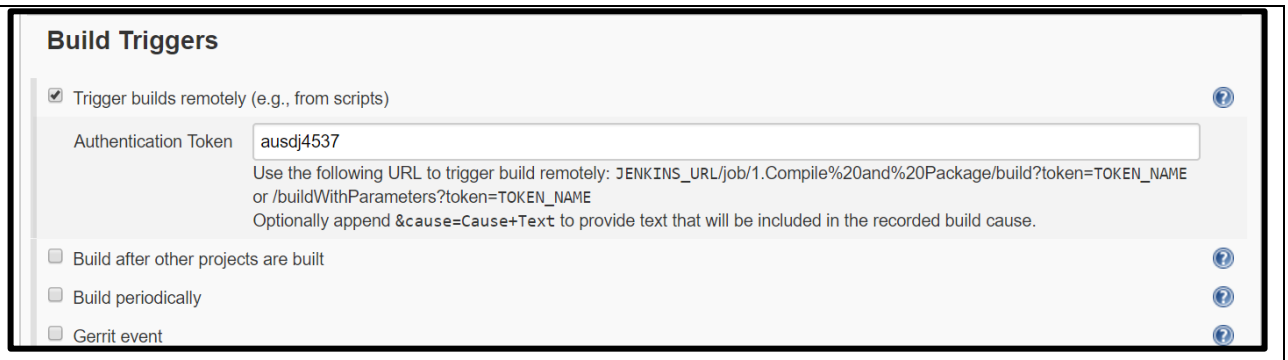


2 After all the jobs are configured, you can execute jobs as per the specified configuration for **0 touch** deployment.

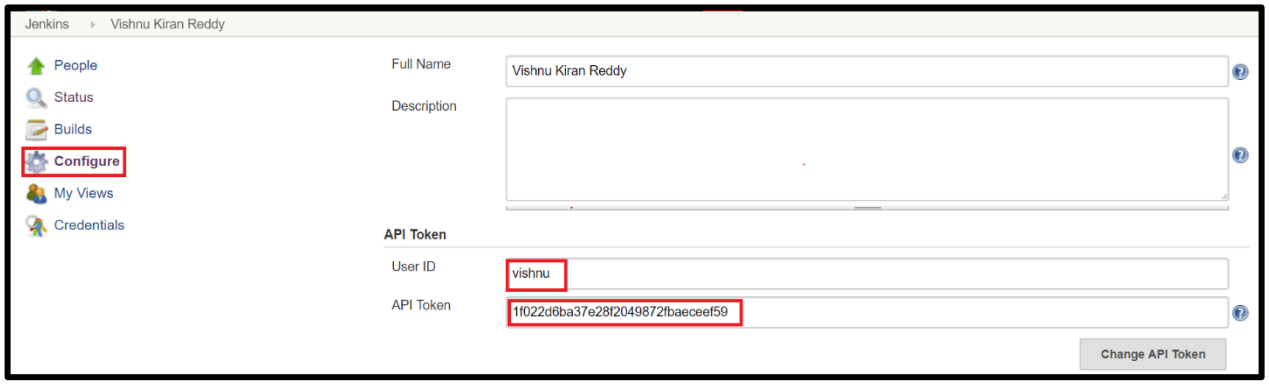
a. Navigate to Jenkins dashboard -> 1.Compile and Package -> Configure

b. Go to Build Triggers section

c. Check the check box **Trigger builds remotely** and enter authentication token (any random text for example:ausdj4537)



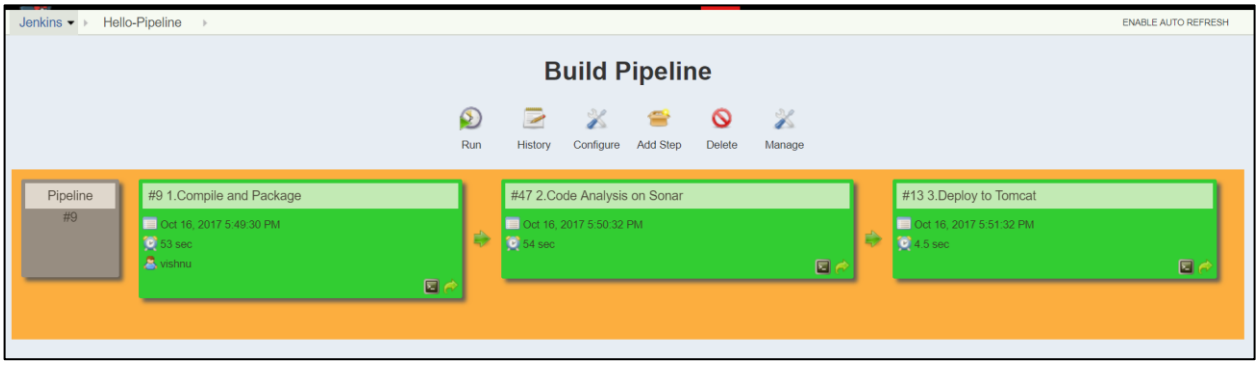
d. Click the **Save** button to save the configuration.
e. Go to Jenkins dashboard. Click the **Full name** present in the top right corner and click **Configure**
f. Click the **Show API Token** button
g. Copy the **Name** and **API Token** and save in notepad.



h. Go to **Git Bash** and go to project repository
   - cd Project
   - cd .git
   - cd hook
   - vi post-commit
     In the vi editor press i and type the below script:

#!/bin/sh
curl --user <jenkinslogin>:<API Token> http://localhost:8082/job/MyJob/build?token= ausdj4537
echo "jenkins from an external script"

i. Type :**wq** and press enter
j. Go to project folder do some changes in the **README** file and push your changes to repository by referring the Exercise 3.1
k. Now you observe the pipeline that is triggered automatically



End of exercises