

Assignments for DevOps Continuous Integration and Continuous Delivery

CONTENTS

CONTENTS	1
Assignments for DevOps Continuous Integration and Continuous Delivery	2
Context	2
Exercise 1: Testing lab set up	3
Exercise 2: Git operations	3
Exercise 3: Creating a project in SonarQube	12
Exercise 4: Using SonarQube with Sonar-Runner	13
Exercise 5: Creating a local repository in Artifactory	16
Build Automation: Maven	17
Exercise 6: Build automation using Maven	17
Continuous Integration: Jenkins	21
Exercise 7: Jenkins Installation & System Configuration	21
Exercise 8: Download the plugins in Jenkins	23
Exercise 9: Creating Central CI pipeline	23
Exercise 10: Copying and Moving Jobs	40
Exercise 11: Creating pipeline view in Jenkins	41
Exercise 12: Configuring Gating Conditions	43
Additional Exercises	47
1. Adding custom rules to SonarQube	47
2. Static program analysis using SonarLint	48
3. Binding SonarQube rules to SonarLint	52

Assignments for DevOps Continuous Integration and Continuous Delivery

Context

This document contains exercises and activities that would be done during the lab practice of DevOps. This would provide hands on experience on the concepts and help the participants create pipelines using open source tool stack.

Application used:

A simple web application using Maven.

Tools that would be used:

- Eclipse – Integrated development environment
- JUnit – Unit testing of code
- Jenkins – Continuous integration server
- Git – Source code management
- Jenkins – Build Automation
- SonarQube – Source code quality management
- JaCoCo – Code coverage

NOTE:

- For performing the exercises below, participants must have the setup of the tools mentioned.
- The screenshots provided are illustrative, based on your system setup, the contents may change.

Exercise 1: Testing lab set up

Step 1: Open the System Environment Variables by right-click on This PC → Properties → Advanced System Settings → Environment Variables...

Add the following as System Variables if not added already:

- JAVA_HOME = path to jdk folder (C:\Program Files\Java)
- M2_HOME = path to maven folder (C:\Program Files\Maven)
- PATH = add "%JAVA_HOME%/bin; %M2_HOME%\bin; <path to sonar-runner>\sonar-runner-2.4\bin;" to the existing path variables

Step 2: Start all the installed tools by executing the appropriate batch file.

Step 3: you can ensure all are working by accessing tools user interface with below mentioned URL's and credentials

1. <http://localhost:9000> – SonarQube [admin/admin]
2. <http://localhost:8080> – Tomcat [tomcat\secret]
3. <http://localhost:8064/jenkins> – Jenkins

Summary: You have tested lab setup for the forthcoming exercises for Jenkins.

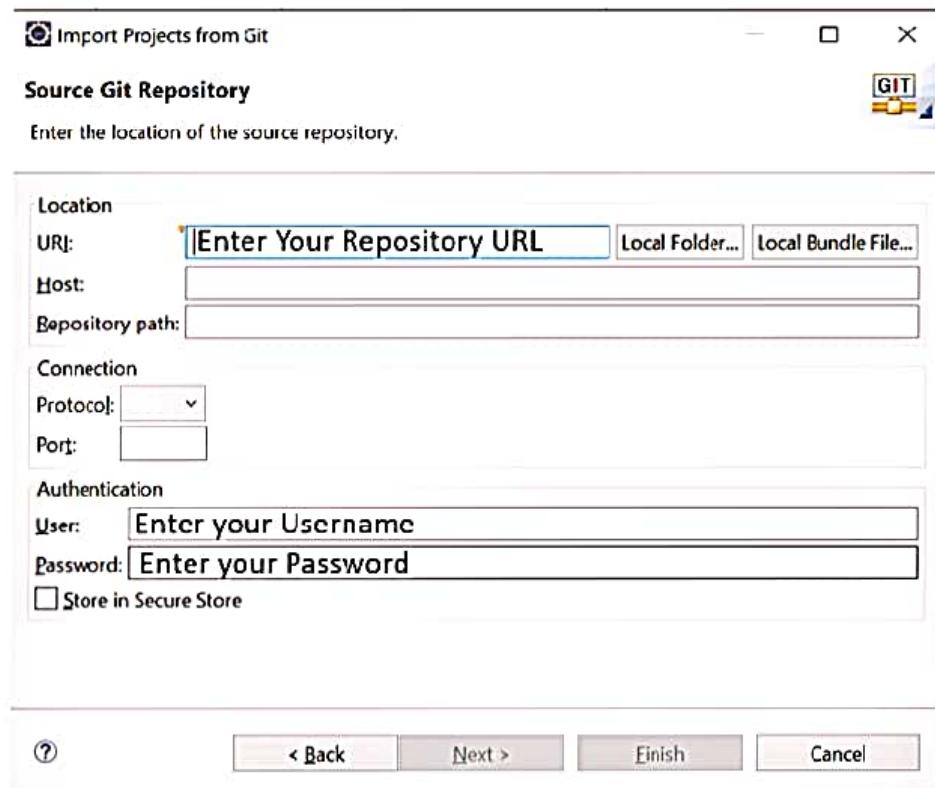
Exercise 2: Git operations

Objective: Perform the basic operations on Git repositories using EGit (Eclipse plug-in)

Pulling code from Git repository

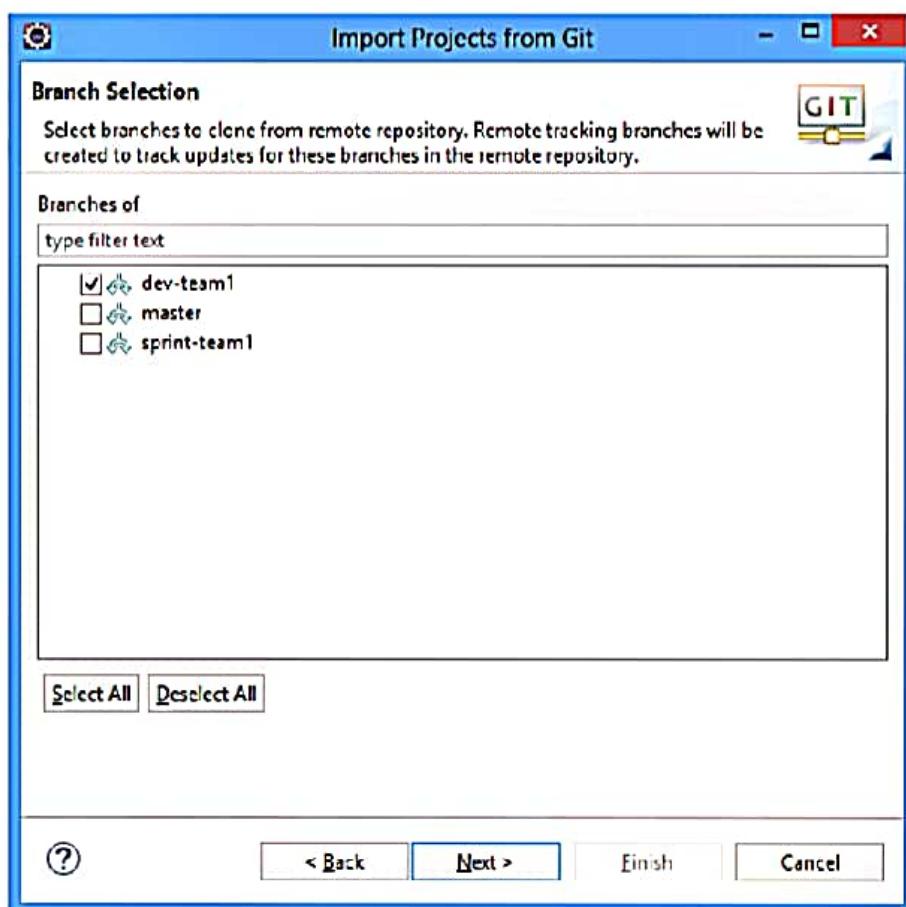
Step 1: Go to Eclipse → file → import → Git → Projects from Git → Clone URI → enter central repository details → use credentials.

Assignments for DevOps Continuous Integration and Continuous Delivery

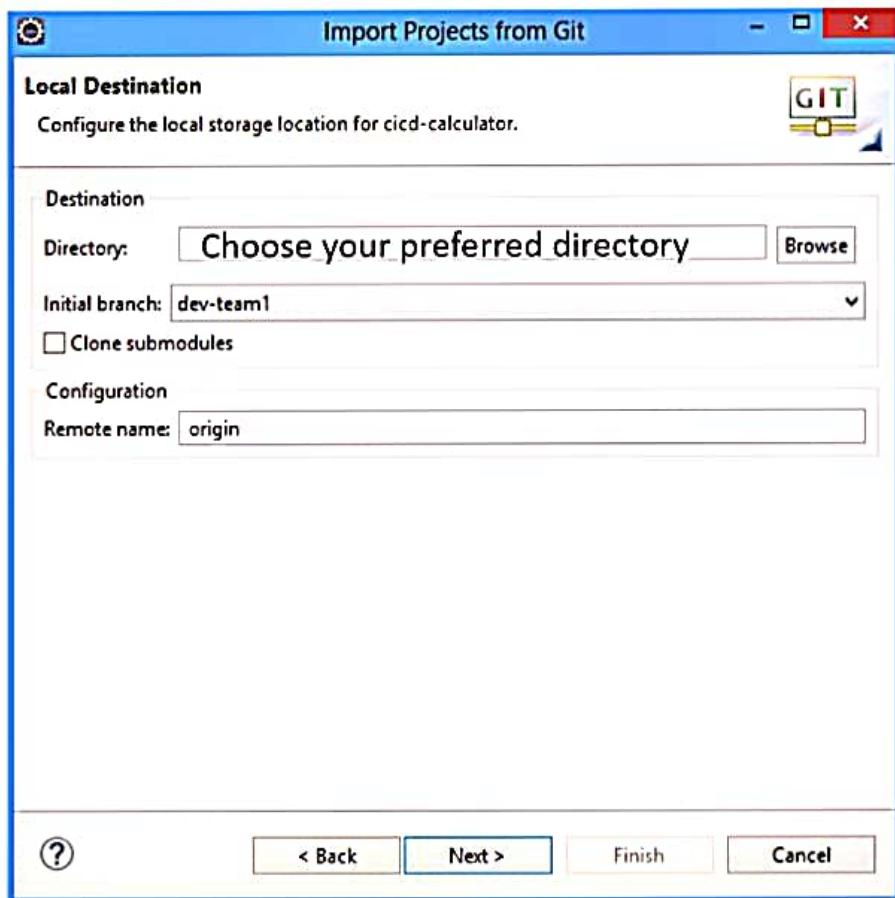


Click Next-> and choose branch (sample shown below)

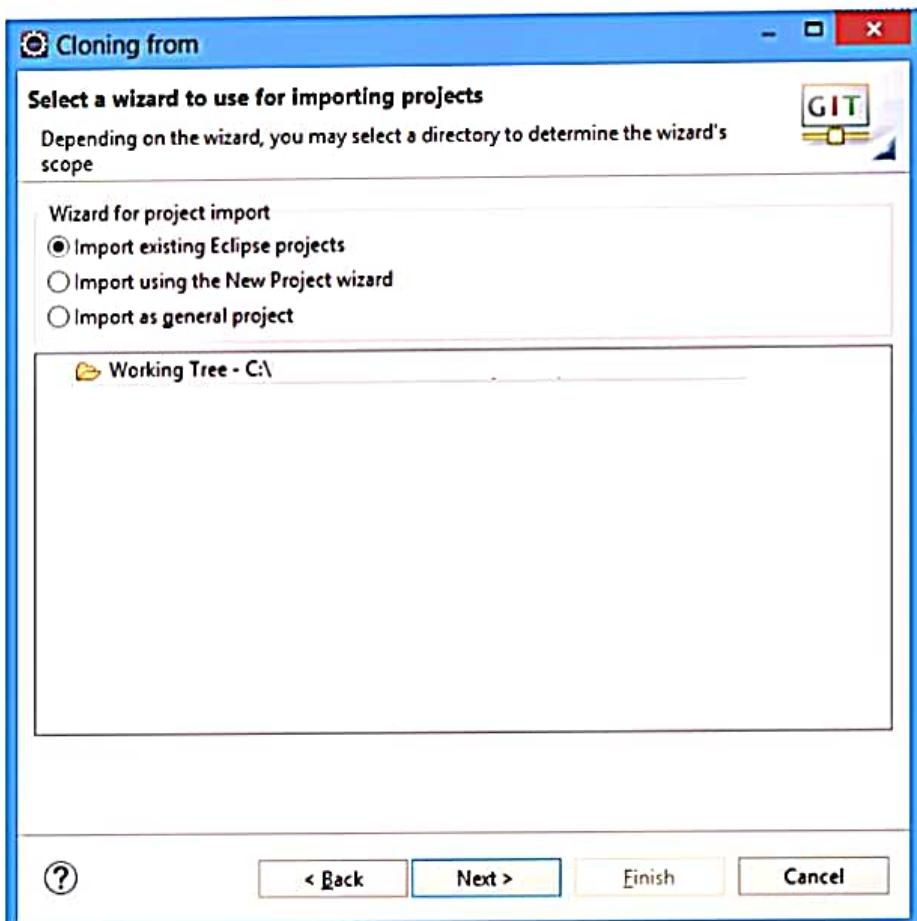
Assignments for DevOps Continuous Integration and Continuous Delivery



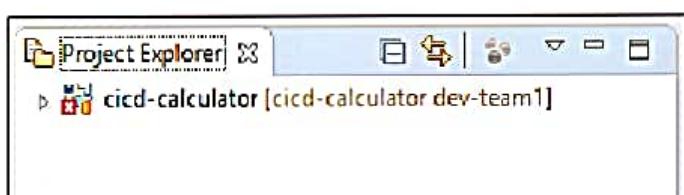
-> Click next> browse directory to keep local repository as shown below->



-> Next-> Choose first option as shown below-> click Next->



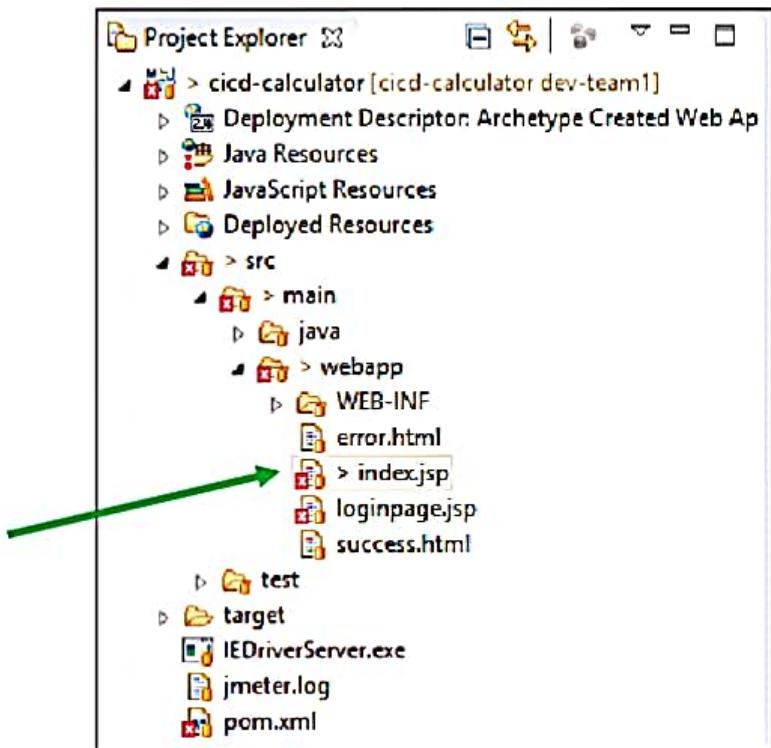
→ Click Next → click Finish → you can observe project explorer view with project cloned as shown below →. The name of the project is JNTU_Calc_Application in your case.



Commit Changes from Eclipse to Local Git Repository

Step 1: Expand the project cicd-calculator → go to `src\main\webapp\index.jsp` → make some changes to the code and save the changes → you can observe ">" symbol on project and edited source file as shown below.

Assignments for DevOps Continuous Integration and Continuous Delivery



Step 2: Right click on project->team->commit->drag the files from Unstaged Changes to Staged Changes ->Enter appropriate comments for the commit as shown below->



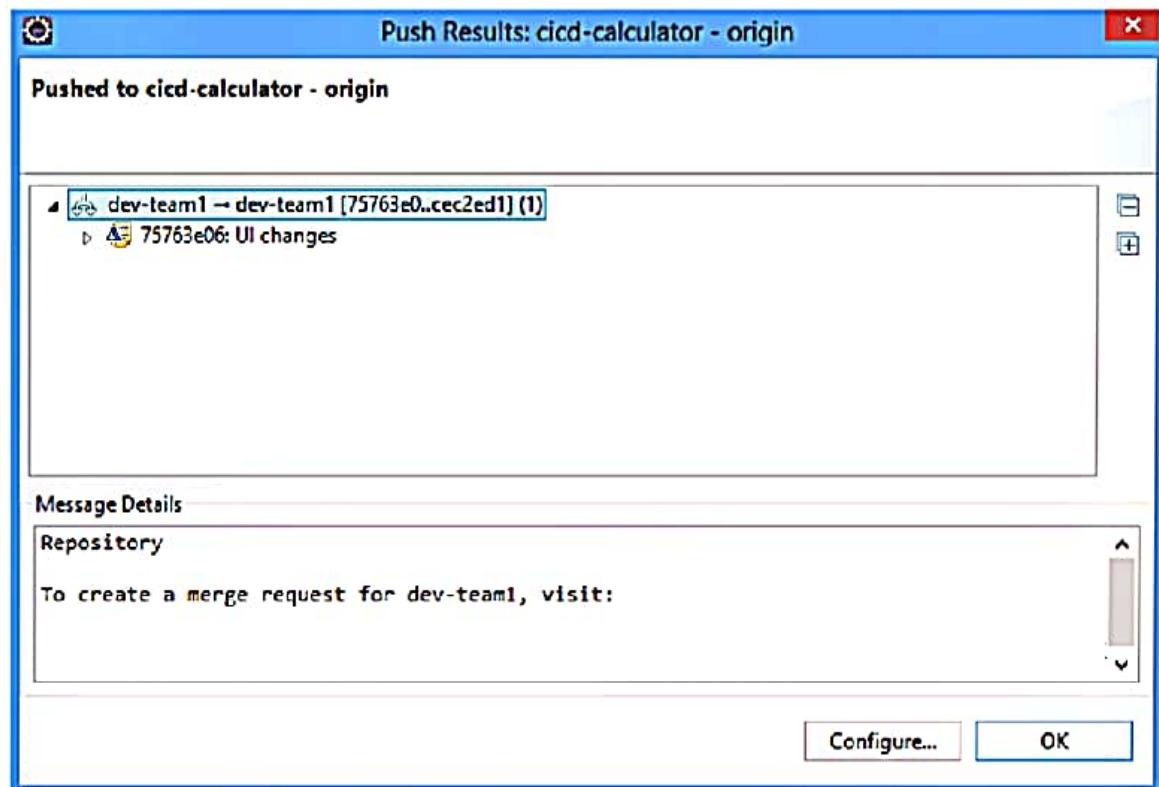
-> click on Commit->you can observe the ">" symbol disappearing.

Note: similar way, we can commit multiple individual changes to local repository using commit option.

Push Changes from Local Git Repository (associated with Eclipse) to Remote Git Repository

Step 1: Right click on project->team-> click on Push to upstream -> you can see changes successfully pushed from local repository to central repository as shown below.

Note: Your repository name and number may be different, this is an illustration.



Note: if push results to "non fast forward" rejection, perform below mentioned operations in sequence.(This is due to the fact that there has been more changes made possibly by other developers to the central Git repository and hence those changes need to be merged before committing the changes)

Fetch from upstream->merge with local branch -> resolve if there are any merge conflicts-> commit-> push.

You can observe the revision history using project->team->"Show In History" as shown below.

Assignments for DevOps Continuous Integration and Continuous Delivery

Project: cicd-calculator [cicd-calculator]				
Id	Message	Author	Authored Date	Committer
75763e0650c622ffcd560f9cc11c09871ff04539	[dev-team1] origin/dev-team1 [HEAD] UI changes			
cec2ed1	removed extra html tag			
7809083	Initial commit			

commit 75763e0650c622ffcd560f9cc11c09871ff04539
Author:
Committer:
Parent: [cec2ed10c10f8e281091991de1327eb1ff93a9f14](#) (removed extra html tag)
Branches: [dev-team1](#), [origin/dev-team1](#)

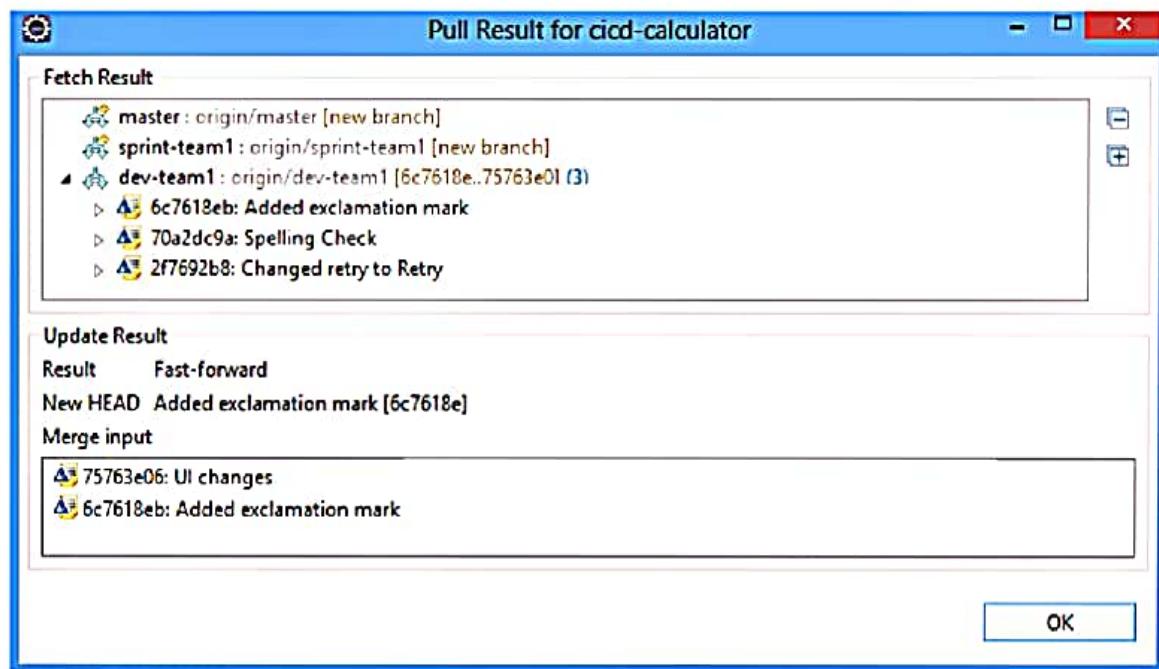
UI changes

src/main/webapp/index.jsp

Pull Changes from Remote Git Repository to Local Git Repository (associated with Eclipse)

Assumption: A team member has committed three changes to remote repository.

Step 1: Right click on project->team->Pull->enter credentials if required-> you can observe fetch result and merge input see as shown below



Note: Pull perform two actions in sequence, Fetch and Merge. So if there are any merge conflicts after pull operation, you have to resolve and commit again.

You can observe the updated revision history using project->team->"Show in History".

Pull the project from remote Git repository to local Eclipse work space using EGit.
Practice the following commands-

- a. Commit and push operations
- b. Fetch operation
- c. Merge operation with and without conflicts. When there is a conflict, resolve the conflict and push the code back to the branch provided.

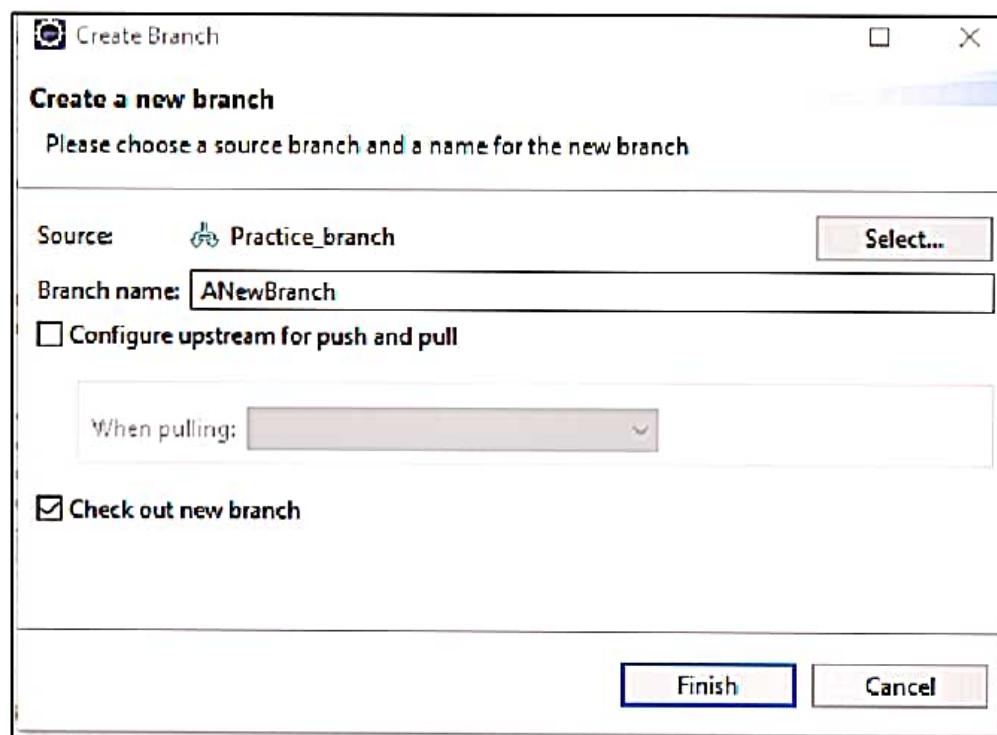
Note: Perform commit and push operations on remote Git repository by making some changes to source code. You can use the repository provided in the earlier demo for completing this exercise.

Creating branches on local repository from eclipse via e-git plugin:

Step 1: Right click on the Project in the Project Explorer and go to Teams -> Switch to -> New Branch...

Step 2: The parent branch will by default be that branch that is currently open in Eclipse, you can also change it by clicking on the Select option.

Step 3: Name the New Branch and check on the Check out as new branch if you would like to switch to the new branch as well Click on Finish.



Summary: You have learnt to do basic operations with Git related to version control in this exercise.

Exercise 3: Creating a project in SonarQube

Objective: Understand creation of project in SonarQube

Step 1: Go to SonarQube URL and login with credentials: admin: password

Step 2: Click on the manually option on the SonarQube dashboard as shown in the screenshot given below.

Are you just testing or have an advanced use-case? Create a project manually.



Step 3: Enter the project display name and project key as calculator as shown in the screenshot given below and click Set Up.

A screenshot of the "Create a project" form. The form has a light gray background. At the top, there is a message: "All fields marked with * are required". Below this, there are two input fields: "Project display name *" and "Project key *". Both fields contain the value "calculator". Under each field, there is a brief description. After the "Project key *" field, there is a note about the allowed characters. At the bottom of the form, there is a blue "Set Up" button.

Step 4: Click on Locally option as shown in the screenshot given below.

Are you just testing or have an advanced use-case? Analyze your project locally.



Step 5: Click on the Generate option as shown in the screenshot given below.

Assignments for DevOps Continuous Integration and Continuous Delivery



Step 6: You can see the token generated as shown in the screenshot given below. Click on continue.

Note: Copy the token and save it somewhere on your system. It cannot be retrieved later.



Step 7: Select Maven under the run analysis on your project as shown in the screenshot given below.

2 Run analysis on your project

What option best describes your build?

Maven Gradle .NET Other (for JS, TS, Go, Python, PHP, ...)

Step 8: Paste the copied token in the pom.xml within the <sonar.login> tag as shown in the screenshot given below.

<sonar.login>Place the token here</sonar.login>

Summary: You have learnt to create a project in sonarqube in this exercise.

Exercise 4: Using Sonarqube with Sonar-Runner

Objective: Understand running of Sonarqube using Sonar-Runner command-line tool

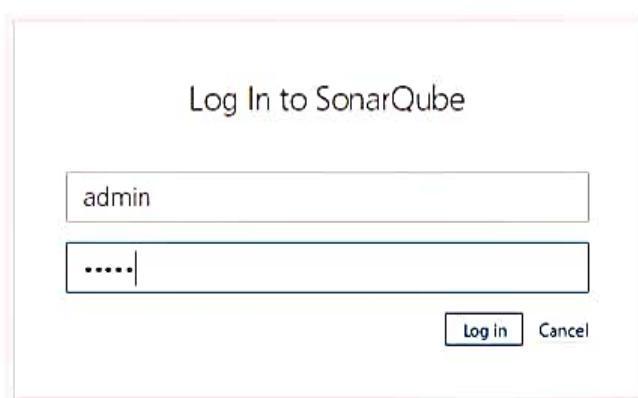
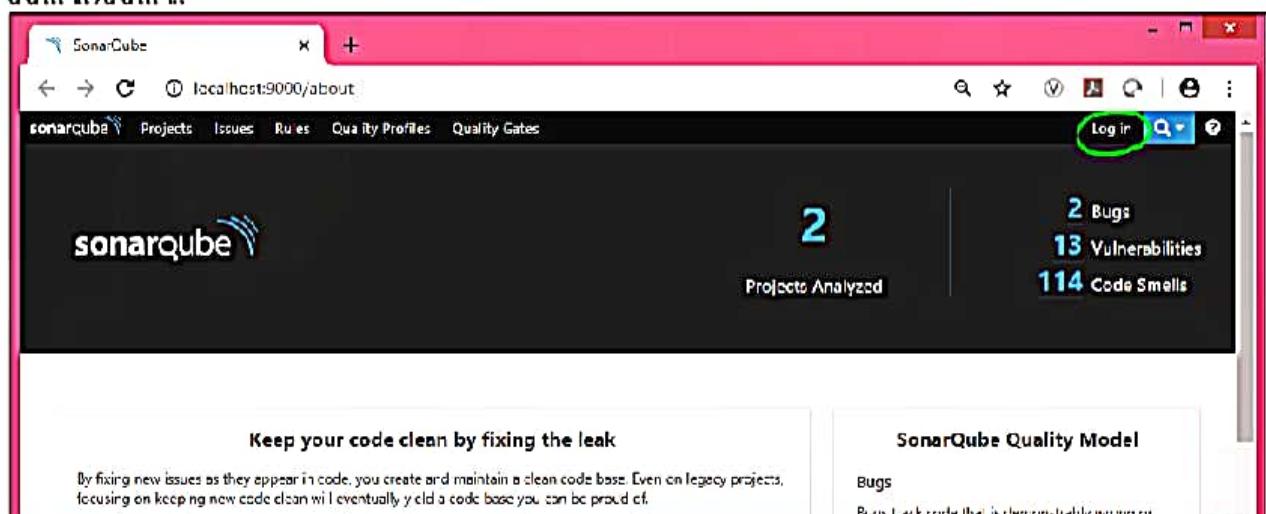
Step 1: Start Sonar server by using the appropriate bat file.

Once started, you should see success message in console window:

Assignments for DevOps Continuous Integration and Continuous Delivery

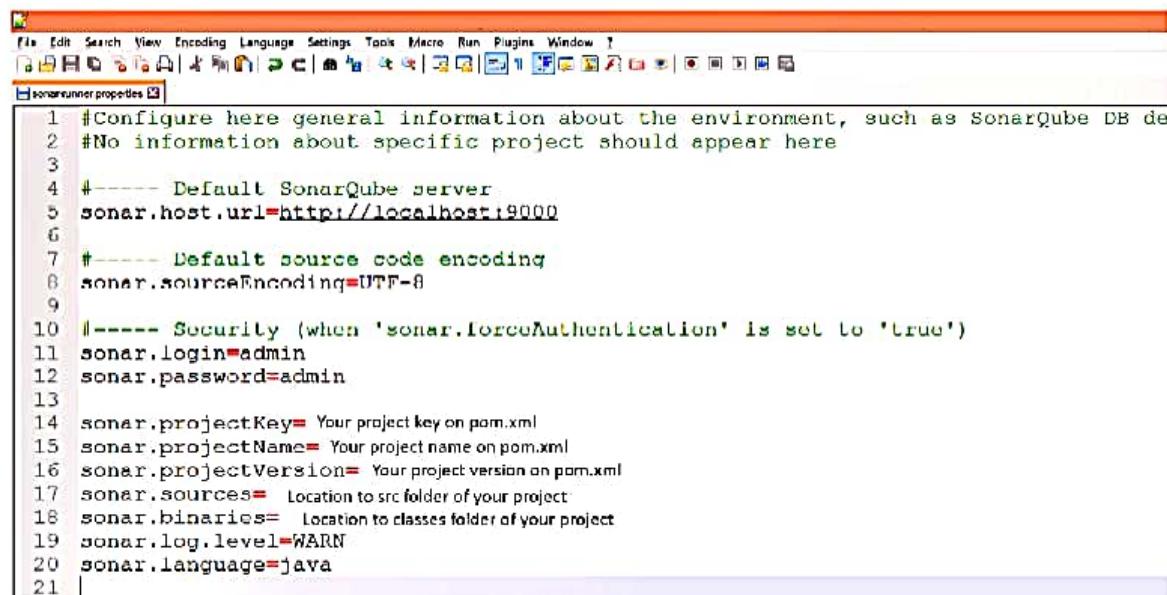
```
 SonarQube
jvm 1 | 2019.01.20 00:02:57 INFO app|[o.s.p.m.Monitor] Process[web] is up
jvm 1 | 2019.01.20 00:02:57 INFO app|[o.s.p.m.JavaProcessLauncher] Launch process[ce]
e\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Xmx512m -Xms128m -XX:+HeapDumpO
sonarqube-6.2\temp -javaagent:C:\Program Files\Java\jdk1.8.0_141\jre\lib\manag
b/server/*;./lib/ce/*;C:\          \sonarqube-6.2\lib\jdbc\h2\h2-1.3.176.jar org.sonar.c
e-6.2\temp\sq-process822334252546262056properties
jvm 1 | 2019.01.20 00:03:12 INFO app|[o.s.p.m.Monitor] Process[ce] is up
jvm 1 | 2019.01.20 00:03:12 INFO app|[o.s.application.App] SonarQube is up
```

Go to SonarQube server dashboard at <http://localhost:9000> and login using admin/admin



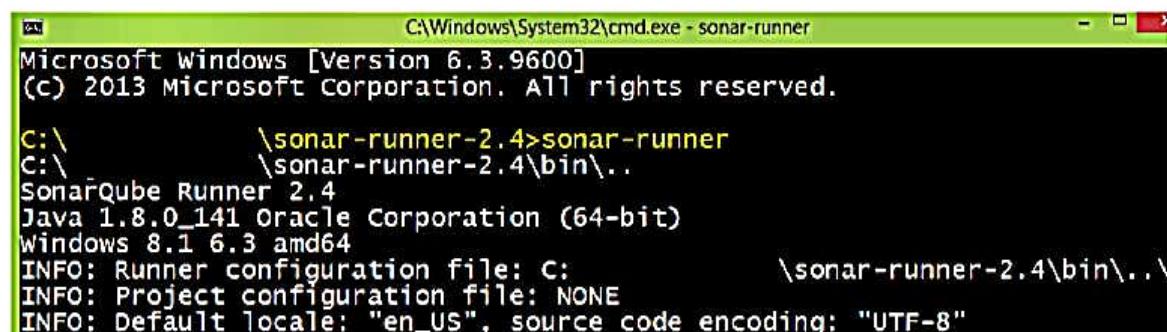
Step 2: Go to conf folder of Sonar Runner. Make the following changes based on the name of the Project provided to you and the path where it exists in your system:

Assignments for DevOps Continuous Integration and Continuous Delivery



```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window 
sonarrunner.properties
1 #Configure here general information about the environment, such as SonarQube DB de
2 #No information about specific project should appear here
3
4 #----- Default SonarQube server
5 sonar.host.url=http://localhost:9000
6
7 #----- Default source code encoding
8 sonar.sourceEncoding=UTF-8
9
10 #----- Security (when 'sonar.forceAuthentication' is set to 'true')
11 sonar.login=admin
12 sonar.password=admin
13
14 sonar.projectKey= Your project key on pom.xml
15 sonar.projectName= Your project name on pom.xml
16 sonar.projectVersion= Your project version on pom.xml
17 sonar.sources= Location to src folder of your project
18 sonar.binaries= Location to classes folder of your project
19 sonar.log.level=WARN
20 sonar.language=java
21 |
```

Open command prompt inside the projectsrc folder in File Explorer and run the following command:



```
C:\Windows\System32\cmd.exe - sonar-runner
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

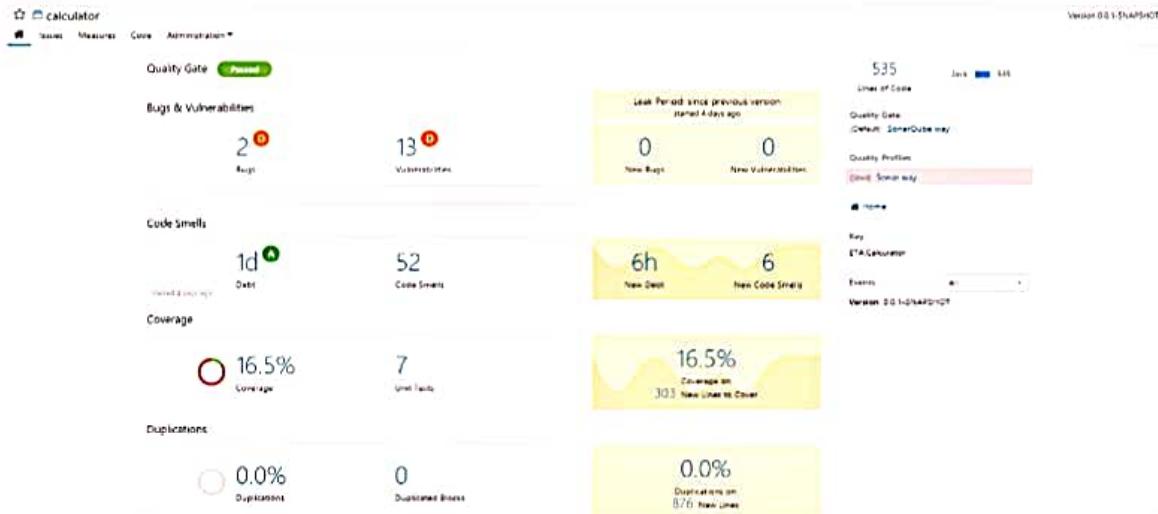
C:\          \sonar-runner-2.4>sonar-runner
C:\          \sonar-runner-2.4\bin\..
SonarQube Runner 2.4
Java 1.8.0_141 Oracle Corporation (64-bit)
Windows 8.1 6.3 amd64
INFO: Runner configuration file: C:          \sonar-runner-2.4\bin\..\\
INFO: Project configuration file: NONE
INFO: Default locale: "en_US", source code encoding: "UTF-8"
```

Once execution is successful

- Observe the static code analysis report and critical issues on SonarQube dashboard. Select your project name from list of projects to view the latest report
- Resolve the technical issues in code and rerun the Maven build. Notice the changes to the technicaldebt value.

Run the build file and observe the results.

Assignments for DevOps Continuous Integration and Continuous Delivery



Summary of this Exercise:

You have learnt to observe the results of static code analysis using Sonarqube

Exercise 5: Creating a local repository in Artifactory

Objective: Understand creation of local repository in Artifactory

Step 1: Go to Artifactory URL and login with credentials: admin: Password1!

Step 2: Go to Administration → Repositories → Repositories → Add repositories → Local Repository.

Step 3: Select the package type as Maven

Step 4: To add the repository key, go to pom.xml and copy the <name> tag value (Calc_Dev_Snapshot) as shown in the screenshot given below.

```
<distributionManagement>
    <repository>
        <id>artifactory</id>
        <name>Calc_Dev_Snapshot</name>
        <url>http://localhost:8081/artifactory/Calc_Dev_Snapshot</url>
    </repository>
</distributionManagement>
```

Step 5: Click on Create Local Repository.

Step 6: You can view the binaries stored in the artifactory under Application → Artifactory → Packages.

Summary: You have learnt to create a local repository in artifactory in this exercise.

Build Automation: Maven

Exercise 6: Build automation using Maven

Objective: Understand build automation by writing a script in Maven with goals to invoke activities in a CI pipeline.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ETA</groupId>
    <artifactId>Calculator</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>calculator</name>
    <url>http://calculator</url>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.6.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <finalName>calculator</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.1.1</version>
                <configuration>
                    <archive>
                        <manifestEntries>
                            <version>${project.version}</version>
                        </manifestEntries>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>

```

Assignments for DevOps Continuous Integration and Continuous Delivery

```
        </manifestEntries>
    </archive>
</configuration>
</plugin>
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>3.2</version>
</plugin>
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.7.9</version>
    <executions>
        <execution>
            <id>default-prepare-agent</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>default-report</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
        <execution>
            <id>default-check</id>
            <goals>
                <goal>check</goal>
            </goals>
            <configuration>
                <rules>
                    <!-- implementation is
needed only for Maven 2 -->
                    <rule
implementation="org.jacoco.maven.RuleConfiguration">
<element>BUNDLE</element>
                    <limits>
                        <!--
implementation is needed only for Maven 2 -->
                        <limit
implementation="org.jacoco.report.check.Limit">
<counter>COMPLEXITY</counter>
<value>COVEREDRATIO</value>
<minimum>0.10</minimum>
                        </limit>
                    </limits>
                </rule>
            </rules>
        
```

Assignments for DevOps Continuous Integration and Continuous Delivery

```
                </configuration>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>

<profiles>
    <profile>
        <id>ut</id>
        <build>
            <plugins>
                <plugin>

                    <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-surefire-
plugin</artifactId>
                            <configuration>
                                <includes>

<include>**/Calculaterut.java</include>

                                </includes>
                            </configuration>
                        </plugin>
                    </plugins>
                </build>
            </profile>
        <profile>
            <id>it</id>
            <build>
                <plugins>
                    <plugin>

                        <groupId>org.apache.maven.plugins</groupId>
                            <artifactId>maven-surefire-
plugin</artifactId>
                                <configuration>
                                    <includes>

<include>**/CalculatorIT.java</include>
                                </includes>
                            </configuration>
                        </plugin>
                    </plugins>
                </build>
            </profile>
        <profile>
            <id>pt</id>
            <build>
                <plugins>
                    <plugin>
```

Assignments for DevOps Continuous Integration and Continuous Delivery

```
<groupId>com.lazerycode.jmeter</groupId>
            <artifactId>jmeter-maven-
plugin</artifactId>
            <version>2.4.0</version>
            <executions>
                <execution>
                    <id>jmeter-tests</id>
                    <phase>test</phase>
                    <goals>
                        <goal>jmeter</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
</profile>
</profiles>

</project>

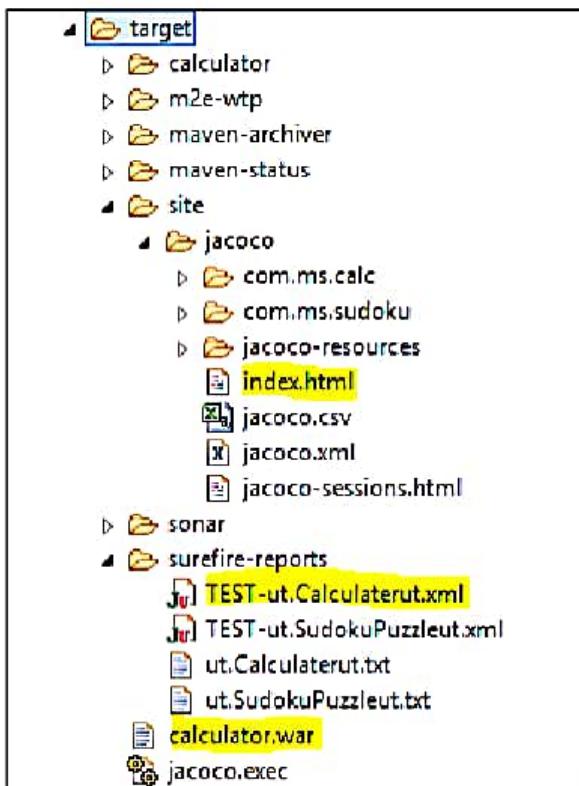
<!-- -->
```

Once you run the pom.xml in the order "clean compile test jacoco:reportsonar:sonar war:war", you can see the output on console as shown below.

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 48.720 s
[INFO] Finished at:
[INFO] Final Memory: 43M/351M
[INFO] -----
```

You can also find the results of the maven goals executed in the target folder of your project

1. JUnit test reports in xml and txt surefire-reports folder
2. Jacoco code coverage reports in html and xml Under site/jacoco/
3. War file



Summary of this Exercise: You have learnt to use Maven tool for build automation.

Continuous Integration: Jenkins

Exercise 7: Jenkins Installation & System Configuration

Objective: Configure Jenkins for CI

Note:

You can install Jenkins in two ways on Windows –

- Install Jenkins as a Windows service
- Use webserver with a servlet container like GlassFish or Tomcat, and then deploy Jenkins.war to it

We have used the first approach in this exercise.

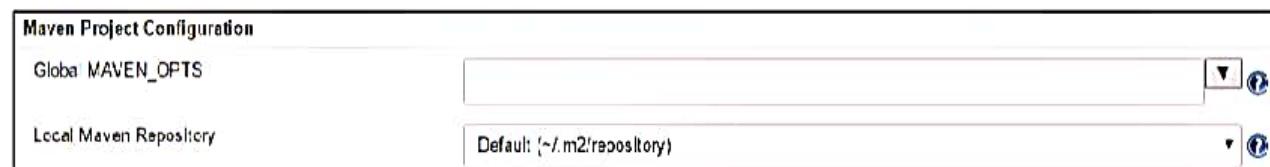
Configuring Jenkins

Step 1: Start the Jenkins server and enter URL <http://localhost:8064/jenkins> in browser

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 2: Go to  Manage Jenkins →  Configure System
Configure global settings and paths.

Step 3: You can observe Jenkins Maven integration as shown below:



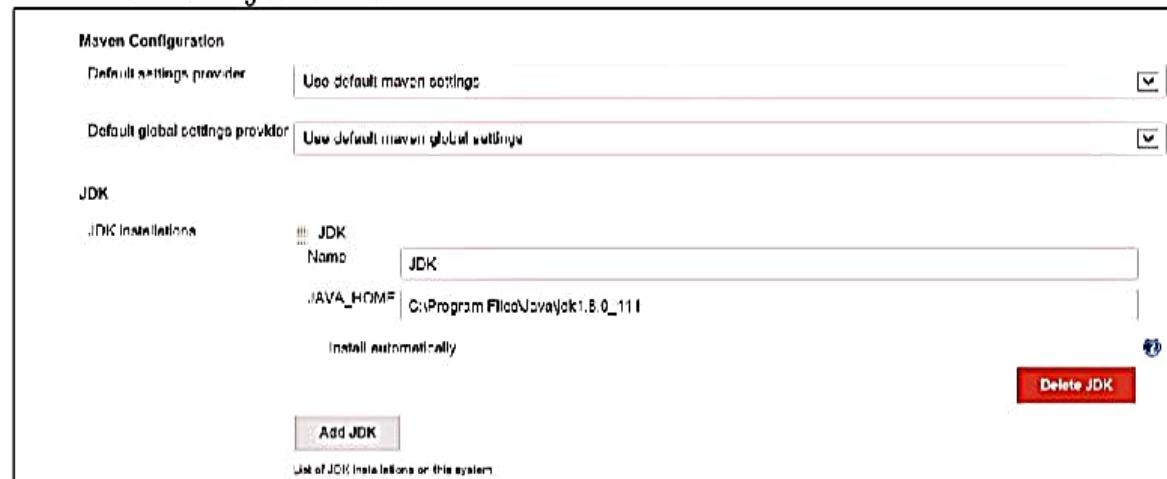
Maven Project Configuration

Global MAVEN_OPTS

Local Maven Repository Default (~/.m2/repository)

Additional configurations

- Provide the tool configuration (JDK, Maven) in the Manage Jenkins → Global Tool Configuration tab



Maven Configuration

Default settings provider Use default maven settings

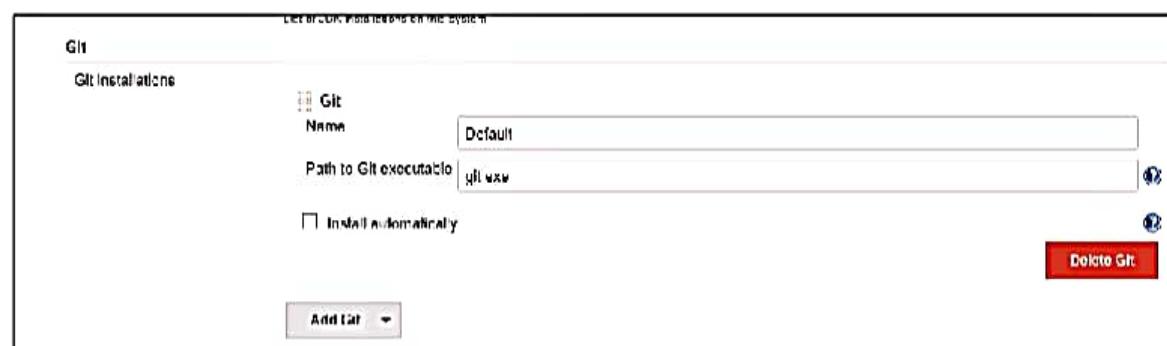
Default global settings provider Use default maven global settings

JDK

JDK installations

 Name	JDK
/JAVA_HOME	C:\Program Files\Java\jdk1.8.0_111
<input type="checkbox"/> Install automatically	
<input type="button" value="Delete JDK"/>	

List of JDK installations on this system



Git

Git installations

 Name	Default
Path to Git executable	git.exe
<input type="checkbox"/> Install automatically	
<input type="button" value="Delete Git"/>	

- Go to global properties (Manage Jenkins→Configure system→Global properties→environment variables) and set JAVA_HOME and M2_HOME to the respective machine path

Summary of this Exercise:

You have learnt to configure Jenkins.

Exercise 8: Download the plugins in Jenkins

Objective: Download the plugins in Jenkins

Step 1: Go to Jenkins URL

Step 2: Go to Manage Jenkins → Manage Plugins from the Jenkins dashboard

Step 3: Under the available tab of plugins manager search for the copy artifact plugin and check the check box as shown in the screenshot given below.

Plugin Manager

The screenshot shows the Jenkins Plugin Manager interface. At the top, there's a search bar with the text 'copy artifact'. Below the search bar, there are tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. The 'Available' tab displays a list of plugins. One plugin, 'Copy Artifact 1.46.1', is highlighted. It has a checked checkbox next to 'Build Parameters'. To the right of the plugin name, it says 'Released' and '2 mo 2 days ago'. Below the plugin list, there are two main action buttons: 'Install without restart' and 'Download now and install after restart'. There's also a note 'Update information obtained: 2 days 4 hr ago' and a 'Check now' button.

Step 4: Repeat the step – 3 and select the below mentioned plugins and click on install without restart

- a. JaCoCo
- b. Artifactory
- c. Build pipeline
- d. Deploy to container

Summary: You have learnt to download plugins in Jenkins in this exercise.

Exercise 9: Creating Central CI pipeline

Objective: Creating main line CI pipeline

Create a new Folder item with name “Central_CI” using “New Item” option as shown below. Add jobs of type Freestyle Project for each of tasks needed in the the continuous integration pipeline.

Assignments for DevOps Continuous Integration and Continuous Delivery

Jenkins

Enter an item name

Setup x Required Field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and dependency indicators like configuration.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project

Step 1: Create below mentioned job - Setup

Assignments for DevOps Continuous Integration and Continuous Delivery

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description: code check out

[Plain text] Previous

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

GitHub project

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can run

Source Code Management

None

Git

Repositories

Repository URL: Enter your repo URL

Credentials: Enter Credentials

Branches to build

Branch Specifier (blank for any): Enter your working branch

Repository browser: (Auto)

Additional Behaviours:

Subversion

Assignments for DevOps Continuous Integration and Continuous Delivery

Build Triggers

- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- Build when another project is promoted
- GitHub hook trigger for GITScm polling
- Poll SCM

Schedule: H/2 * * * *

Ignore post-commit hooks

Build Environment

Delete workspace before build starts Advanced...

Abort the build if the check fails Save Apply Console Output

Use secure identity or identity Save Apply

With Ant Save Apply

Build

Invoke top level Maven targets X Save Apply

Maven Version: new Advanced...

Goals: clean Advanced...

Add build step ▾

Post-build Actions

Build other projects X Save Apply

Projects to build: Compile Advanced...

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post-build action ▾

Save Apply

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 2: Create below mentioned job - Compile

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description: compiling

[Plain text] Preview

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

[Advanced...](#)

GitHub project

[Edit](#) [Copy](#) [Archive](#)

Save **Apply**

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can be run

[Advanced...](#)

Source Code Management

None

CVS

Subversion

Build Triggers

Save **Apply** (from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

GitHub hook trigger for GITScm polling

Poll SCM

Build Environment

Delete workspace before build starts

[Advanced...](#)

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s)

Assignments for DevOps Continuous Integration and Continuous Delivery

Copy artifacts from another project

Project name: **Setup**

Which build: **Latest successful build**

Stable build only

Artifacts to copy: ****/***

Artifacts not to copy:

Target directory:

Parameter filters:

Flatten directories Optional Fingerprint Artifacts

Save **Apply** **Advanced...**

Invoke top-level Maven targets

Maven Version: **new**

Goals: **compile**

Advanced...

Add build step ▾

Post-build Actions

Archive the artifacts

Files to archive: ****/***

Advanced...

Build other projects

Projects to build: **StaticAnalysis**

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post build action ▾

Save **Apply**

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 3: Create below mentioned job – Unit test

General

Description: unitTest

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

GitHub project

Permissions to Copy Artifact

Promote builds when ...

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can be run

Source Code Management

None

Git

Subversion

Build Triggers

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

GitHub hook trigger for GITScm polling

Poll SCM

Build Environment

Delete workspace before build starts

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s)

With Ant

Assignments for DevOps Continuous Integration and Continuous Delivery

Build

Copy artifacts from another project

Project name: StaticAnalysis
Which build: Latest successful build
 Stable build only

Artifacts to copy: **/*
Artifacts not to copy:
Target directory:
Parameter filters:
 Flatten directories Optional Fingerprint Artifacts
Advanced...

Invoke top level Maven targets

Maven Version: new
Goals: test -Put
Advanced...

Add build step ▾

Post-build Actions

Archive the artifacts

Files to archive: **/*
Advanced...

Build other projects

Projects to build: CodeCoverage
 Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post-build action ▾

Save Apply

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 4: Create below mentioned job - Code coverage

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description: codeCoverage

[Plain text] Preview

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

[Advanced...](#)

GitHub project

Permission to Copy Artifact

Promote builds when ...

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can be run

[Advanced...](#)

Source Code Management

None

Git

Subversion

[Advanced...](#)

Build Triggers

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

GitHub hook trigger for GIT SCM polling

Poll SCM

[Advanced...](#)

Build Environment

Delete workspace before build starts

[Advanced...](#)

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s)

With Ant

[Advanced...](#)

Assignments for DevOps Continuous Integration and Continuous Delivery

Build

Copy artifacts from another project

Project name: UnitTest
Which build: Latest successful build
 Stable build only

Artifacts to copy: **/*
Artifacts not to copy:
Target directory:
Parameter filters:
 Flatten directories, Optional, Fingerprint Artifacts
Advanced...

Invoke top-level Maven targets

Maven Version: new
Goals: jacoco:report
Advanced...

Add build step ▾

Post-build Actions

Archive the artifacts

Files to archive: **/*
Advanced...

Build other projects

Projects to build: War
 Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post build action ▾

Save **Apply**

Assignments for DevOps Continuous Integration and Continuous Delivery

Build

Copy artifacts from another project

Project name: **Compile** X 

Which build: **Latest successful build** ▼ 

Stable build only

Artifacts to copy: ****/*** 

Artifacts not to copy: 

Target directory: 

Parameter filters: 

Flatten directories Optional Fingerprint Artifacts Advanced...

Invoke top-level Maven targets

Maven Version: **new** ▼ 

Goals: **sonar:sonar** ▼ 

Advanced...

Add build step ▾

Post-build Actions

Archive the artifacts

Files to archive: ****/*** Advanced...

Build other projects

Projects to build: **UnitTest** X 

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post build action ▾

Save Apply

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 6: Create below mentioned job - war

General | Source Code Management | Build Triggers | Build Environment | Build | Post-build Actions

Description: war

[Plain text] [Preview]

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

[Advanced...](#)

GitHub project

Permission to Copy Artifact

Promote builds when ...

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can be run

[Advanced...](#)

Source Code Management

None

Git

Subversion

[Advanced...](#)

Build Triggers

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

GitHub hook trigger for GITScm polling

Poll SCM

[Advanced...](#)

Build Environment

Delete workspace before build starts

[Advanced...](#)

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s)

With Ant

[Advanced...](#)

Assignments for DevOps Continuous Integration and Continuous Delivery

Build Environment

Delete workspace before build starts Advanced...

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s) ?

With Ant ?

Build

Copy artifacts from another project

Project name: **CodeCoverage** ?

Which build: **Latest successful build** ▼ ?

Stable build only

Artifacts to copy: ****/*** ?

Artifacts not to copy: ?

Target directory: ?

Parameter filters: ?

Flatten directories Optional Fingerprint Artifacts Advanced...

Invoke top level Maven targets

Maven Version: **new** ▼

Goals: **war war** ▼

Advanced...

Add build step ▾

Post-build Actions

Archive the artifacts

Files to archive: ****/*** Advanced...

Build other projects

Projects to build: **ToArtifactFactory** ?

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Add post-build action ▾

Save Apply

Page generated Jan 19, 2019 12:10:27 AM IST REST

Assignments for DevOps Continuous Integration and Continuous Delivery

Step 7: Create below mentioned job – To Artifactory

General

Description: war

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept

GitHub project

Permission to Copy Artifact

Promote builds when ..

This project is parameterized

Throttle builds

Disable this project

Execute concurrent builds if necessary

Restrict where this project can be run

Source Code Management

None

Git

Subversion

Build Triggers

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

GitHub hook trigger for GITScm polling

Poll SCM

Build Environment

Delete workspace before build starts

Abort the build if it's stuck

Add timestamps to the Console Output

Use secret text(s) or file(s)

With Ant

Assignments for DevOps Continuous Integration and Continuous Delivery

Build Environment

Delete workspace before build starts

Abort the build if it's stuck
 Add timestamps to the Console Output
 Use secret build(s) or file(s)
 View Art

Build

Copy artifacts from another project

Project name: war

Which build: Latest successful build Stable build only

Artifacts to copy: war

Artifacts not to copy:

Target directory:

Parameter filters: Flatten directories Optional Fingerprint Artifacts

Invoke top level Maven targets

Maven Version: new

Goals: deploy

Post-build Actions

Archive the artifacts

Files to archive: war

Build other projects

Projects to build: SmokeTest

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Step 8: Create below mentioned job – SmokeTest

Assignments for DevOps Continuous Integration and Continuous Delivery

General

Description: smoke test

Discard old builds

Strategy: Log Rotation

Days to keep builds: 3
If not empty, build records are only kept up to this number of days.

Max # of builds to keep: 3
If not empty, only up to this number of build records are kept.

Source Code Management

None

CVS

Subversion

Build Triggers

Trigger builds remotely (e.g. from scripts)

Build after other projects are built

Build periodically

Build when another project is promoted

Gerrit hook trigger for Gerrit SCM polling

Post SCM

Build Environment

Delete workspace before build starts

Advanced...

Audit the build if it's stuck

Add timestamp to the Console Output

Use secret texts (or files)

With Art

Build

Copy artifacts from another project

Project name: Web

Which build: Latest successful build

State build only

Artifacts to copy: my

Artifacts not to copy:

Target directory:

Parameter filters:

Full directory Optional Fingerprint Artifacts

Add build step ▾

Assignments for DevOps Continuous Integration and Continuous Delivery



Summary: You learned main line CI pipeline creation.

Exercise 10: Copying and Moving Jobs

I. Copying Jobs:

Step 1: Click on the option of **New Item** from the left panel

Step 2: Name the new job and enter the name of the job you wish to copy below as shown:

if you want to create a new item from other existing, you can use this option.



Copy from

Type to autocomplete

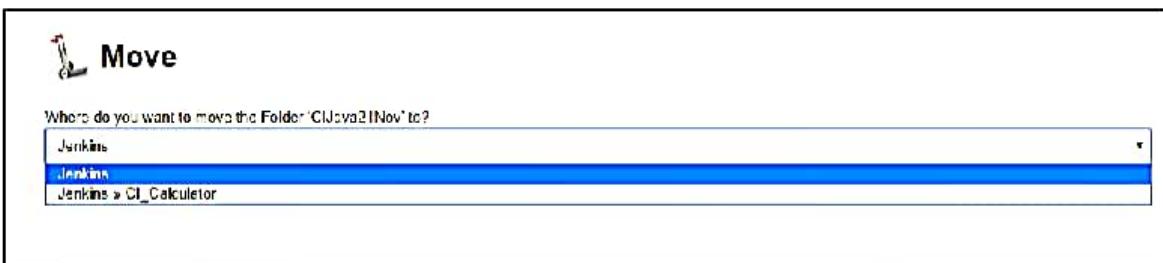
Step 3: Click on **OK** and observe the newly copied job.

II. Moving Jobs:

Step 1: Enter the folder which has the jobs that need to be moved to a new location.

Step 2: Click on the option of **Move** from the left panel

Step 3: Enter the location where you wish to move the jobs to, as shown below.



Step 4: Click on Move and observe the newly moved jobs in that location.
Estimated Time: 20 mins

Summary: You learned how to copy and move jobs on Jenkins.

Exercise 11: Creating pipeline view in Jenkins

Objective: Understand creation of pipeline view in Jenkins

Step 1: Click on the '+' symbol under the Jenkins project folder as shown in the screenshot given below



Step 2: Provide the name for the pipeline in the viewname field and select the Build Pipeline View and click create as shown in the screenshot given below.

Assignments for DevOps Continuous Integration and Continuous Delivery

View name

Type

Build Pipeline View
Shows the jobs in a build pipeline view

Include a global view
Shows the content of a global view.

List View
Shows items in a simple list format. You

Create

Step 3: Select calculator->Setup as the Select Initial Job under the Upstream/downstream config as shown in the screenshot given below and click ok.

Pipeline Flow

Layout

Based on upstream/downstream relationship

This layout mode defines the pipeline structure based on the upstream/downstream relationships between the jobs.

Upstream / downstream config

Select Initial Job ?

Step 4: Execute the job from setup and you can see the pipeline flow as shown in the screenshot given below.



Summary: You learned to create pipeline view in Jenkins.

Exercise 12: Configuring Gating Conditions

Objective: Configure gating conditions in Jenkins.

Apply gates in the tool

Gating condition in Static Analysis job:

Step 1: Start "SonarQube" server if not started earlier.

Step 2: To create quality gate in SonarQube log into SonarQube

(<http://localhost:9000>) as admin (username and password is admin) ->

Quality Gates -> **Create** -> enter "CALQG" as name -> enter "Critical Issues" as metric in "add condition" drop down list and create a rule as shown below and click on add button.

Complexity	Value	is greater than	100	360	Update	Delete
------------	-------	-----------------	-----	-----	--------	--------

Similarly, add another four conditions as shown below

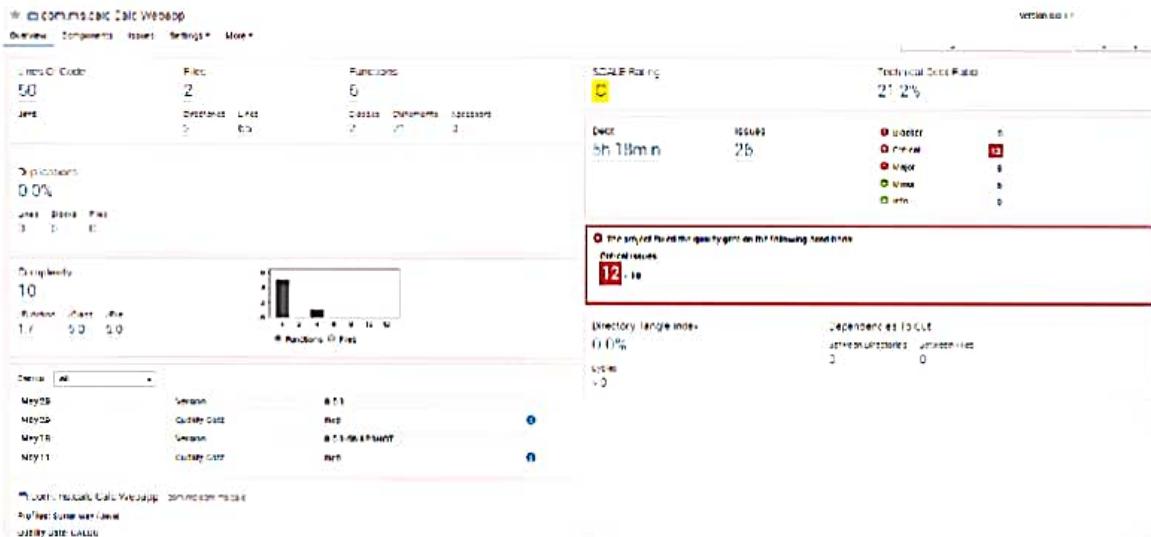
Critical issues	Value	is greater than	10	30	Update	Delete
Duplicated lines	Value	is greater than	50	55	Update	Delete
Major issues	Value	is greater than	10	15	Update	Delete
SCALE Technical Debt Ratio	Value	is greater than	1.2	1.4	Update	Delete

Step 3: To link the "CALQG" with project, use below shown option available in the same page [check the projectname in without section, then that gate applied]

PROJECTS		
With	Without	All
<input type="checkbox"/> com.ms.calc Calc Webapp		

Once static analysis done (by running Maven Target) using SonarQube, you can observe the current analysis results on SonarQube dashboard as shown below.

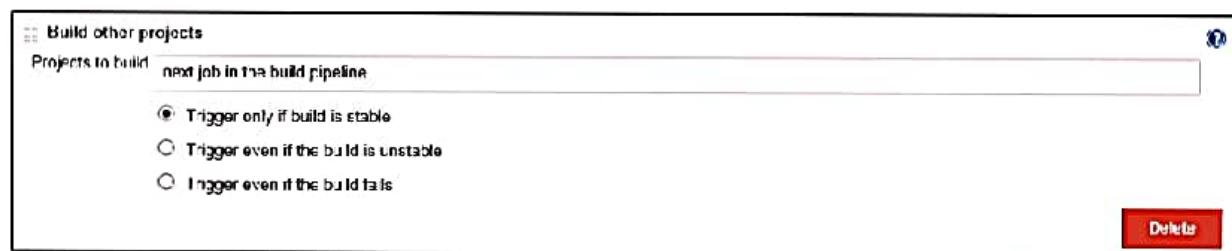
Assignments for DevOps Continuous Integration and Continuous Delivery



Step 4: Enabling build breaker in SonarQube, go to settings->build breaker-> set default to build breakerskip on alertflag option, go to settings -> general settings-> enter value "buildBreaker" in the text field Plugins accepted for Preview and Incremental modes.

Note: this plugin causes Maven target failure, because of quality gate violation

Step 5: Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.



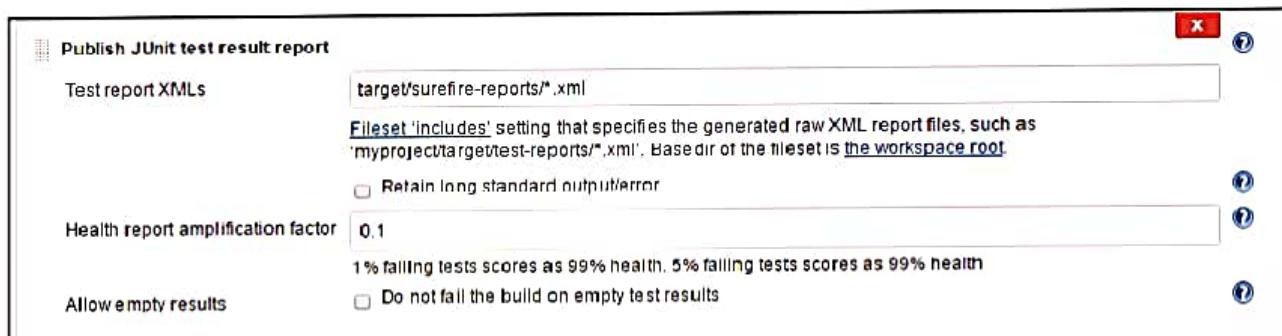
Note: You can configure default quality gates also in SonarQube.

Gating in Jenkins

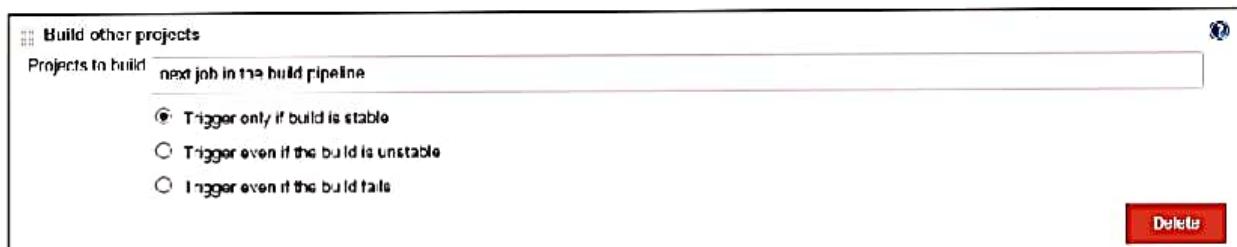
Gating condition in Unit Testing job:

Step 1: You can use unit test results to decide the stability of build by setting up health report amplification factor as shown below.

Assignments for DevOps Continuous Integration and Continuous Delivery



Step 2: Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.



Step 3: Go to the project in Eclipse and edit the junit test class (CalculatorTest.java file under src/main/test/it)

Step 4: Modify any one of the testcases to get fail as shown in the screenshot given below.

```
@Test  
public void testAdd() {  
    assertTrue(CL.doAdd(1, 2) == 4);  
}
```

Step 5: Commit the code and check the Jenkins pipeline for the failure.

Step 6: You can also fail few more testcases and observe the result in the Jenkins pipeline.

Gating condition in Code Coverage job:

Method 1:

Step 1: You can use code coverage results to decide the stability of build as shown below.

Assignments for DevOps Continuous Integration and Continuous Delivery

Record JaCoCo coverage report

Path to exec file (e.g. **/target/**/exec-
**/jacoco.exec)

Inclusions (e.g. **/*class)
Exclusions (e.g. **/*Test.class)

Path to class directories (e.g. **/target/classDir, **/classes)

Path to source directories (e.g. **/src/main/java), Inclusions (e.g. **/java/**/generated/**/gt), Exclusions (e.g. generated/**/java)

Disable display of source files for coverage

Change build status according the thresholds

Always run coverage collection, even if build is FAILED or ABORTED

Instruction	% Branch	% Complexity	% Line	% Method	% Class
♂	0	90	0	90	0
♀	0	0	0	0	0

Fail the build if coverage degrades more than the set thresholds

Instruction	% Branch	% Complexity	% Line	% Method	% Class
♂	0	70	0	80	0

Add post-build actions: |

Step 2: Choose the first option as shown below in Jenkins job configuration to link with downstream job in the pipeline.

Build other projects

Projects to build: next job in the build pipeline

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Delete

Here is a summary of the activities to be done

- a) Open SonarQube (<http://localhost:9000>). Add gating conditions as learnt in the demos. Give values so that you can observe a broken and a smooth build
- b) Trigger a build in Jenkins and observe the gating condition in SonarQube working.
- c) Trigger a build in Jenkins and observe the gating condition in JaCoCo working.
- d) Configure the threshold values for unit testing in Jenkins. Make changes to the unit tests (so that some of them fail) and observe the gating conditions

failing in Jenkins. Correct them and observe that the build becomes stable. (when changes are being made, commit the testcode from Eclipse).

Summary of this Exercise:

You have learnt to apply gating conditions to ensure code quality and tests pass in every build.

Additional Exercises

1. Adding custom rules to SonarQube

Objective: Add custom rules to SonarQube

Requirements: SonarQube 5.6 and above

Step 1: Run the StartSonar.bat  file as admin.

Creating the custom rule

Rule to be created:

Avoid single parameter in methods. Here are the parameters:

1. name: Avoid usage of single parameter
2. Description: This rule expects methods to have at least two parameters
3. This is a coding guideline
4. Priority is MAJOR

Add this rule to SonarQube and check the same using the calculator workspace provided

Hint: Use method.parameters().size() to check the number of parameters in a method

Estimated time: 30 mins

Summary: You have learnt to customize the rules in SonarQube according to your requirement for quality analysis.

2. Static program analysis using SonarLint

Objective: Perform static program analysis during coding using SonarLint plug-in to Eclipse IDE.

SonarLint is a plug-in to an IDE that provides on the fly feedback to developers on new bugs and quality issues injected into their code.

Installing the SonarLint plug-in

Step 1: Go to Eclipse IDE and select a workspace.

Step 2: Import the project [JNTU_Calc_Application](#) provided and switch to JAVA EE perspective.

Step 3: Now open the calculator.java file from 'src/main/java' package.

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The left sidebar shows the Project Explorer with a Java project named 'calculator'. The center area displays the code for 'calculator.java':

```
1 // To be consistent with the naming conventions to be adopted by JavaCommunity
2 package com.eccl;
3
4 // A class to do mathematical operations on numbers
5 // Class name in small letters
6
7 public class calculator {
8
9     //Method to add two numbers
10    //Method name starts with a capital letter
11    public int add(int a, int b) {
12        //Variables declared in capital ones
13        int n = 0;
14        n = a + b;
15        return n;
16    }
17    // else if a == 0 & b == 0
18    // else
19    // else
20    // else
21    // else
22}
```

The right side shows the SonarLint Issues view with 8 errors, 11 warnings, and 0 others. The table lists:

Description	Resource	Path	Location	Line
Cleancpp Dependency Validator Message [Error]				
Java JUnit 4.8 Problems [Errors]				
Java Problems [Errors]				
Manual Configuration Problem [Error]				
Manual Java Compiler Problem [Error]				
Manual Problems [Errors]				
Manual Problems [Errors]				

Working with SonarLint

Step 1: Go to Windows>Show View>Other.

2. Static program analysis using SonarLint

Objective: Perform static program analysis during coding using SonarLint plug-in to Eclipse IDE.

SonarLint is a plug-in to an IDE that provides on the fly feedback to developers on new bugs and quality issues injected into their code.

Installing the SonarLint plug-in

Step 1: Go to Eclipse IDE and select a workspace.

Step 2: Import the project [JNTU_Calc_Application](#) provided and switch to JAVA EE perspective.

Step 3: Now open the calculator.java file from 'src/main/java' package.

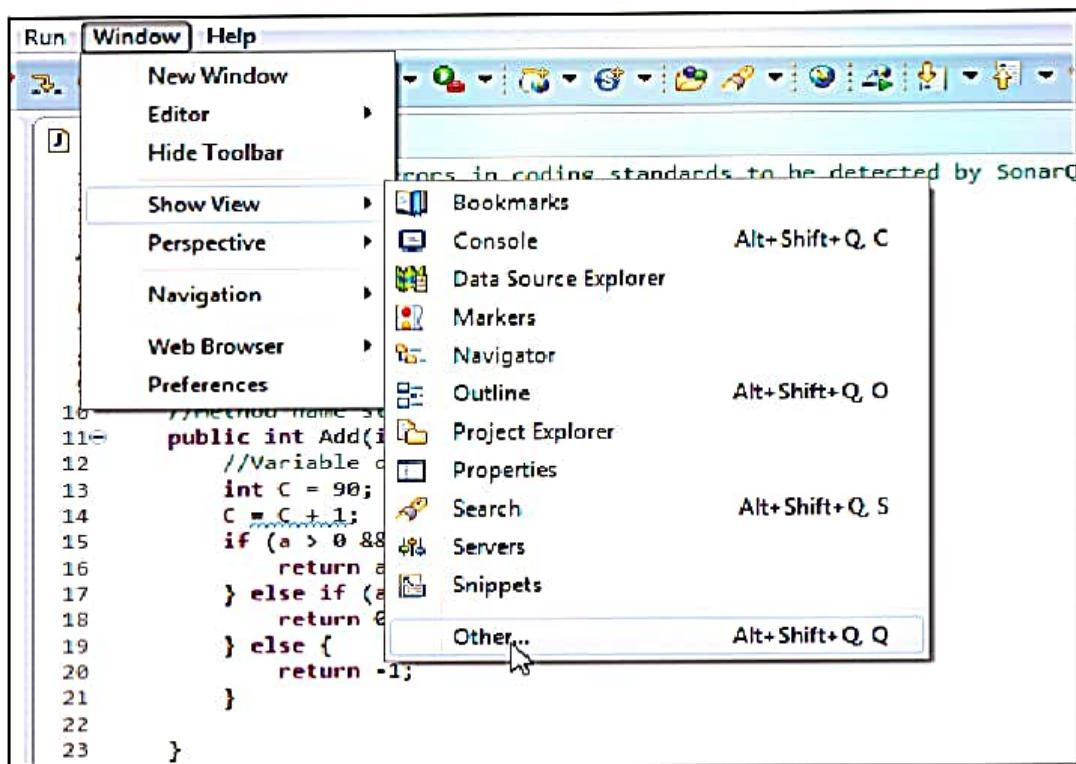
The screenshot shows the Eclipse IDE interface. The left side displays the Project Explorer with a Java project named 'calculator'. The center shows the code editor with the 'calculator.java' file open, containing a simple calculator class with an add method. The right side shows the 'Problems' view, which lists several SonarLint findings, including:

Description	Resource	Path	Location	Line
Classpath Dependency Validator: Missing JBoss Seam API				
Java Build Path: Problem: No items				
Java: Problem: No source				
Java Configuration: Problem: No build path				
Java: Problem: Configuration Problem: No build path				
Java: Problem: Configuration Problem: No build path				

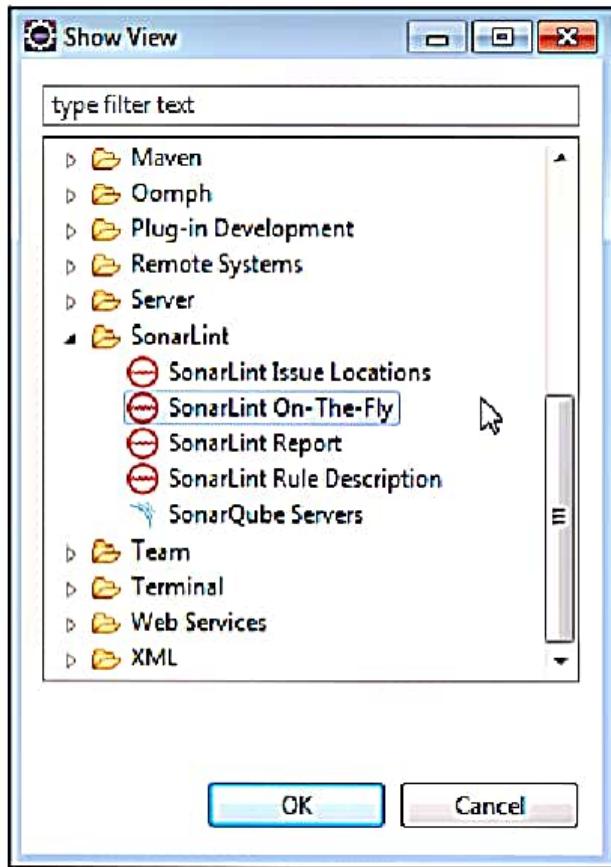
Working with SonarLint

Step 1: Go to Windows > Show View > Other...

Assignments for DevOps Continuous Integration and Continuous Delivery



Here select SonarLint > SonarLint On-The-fly.

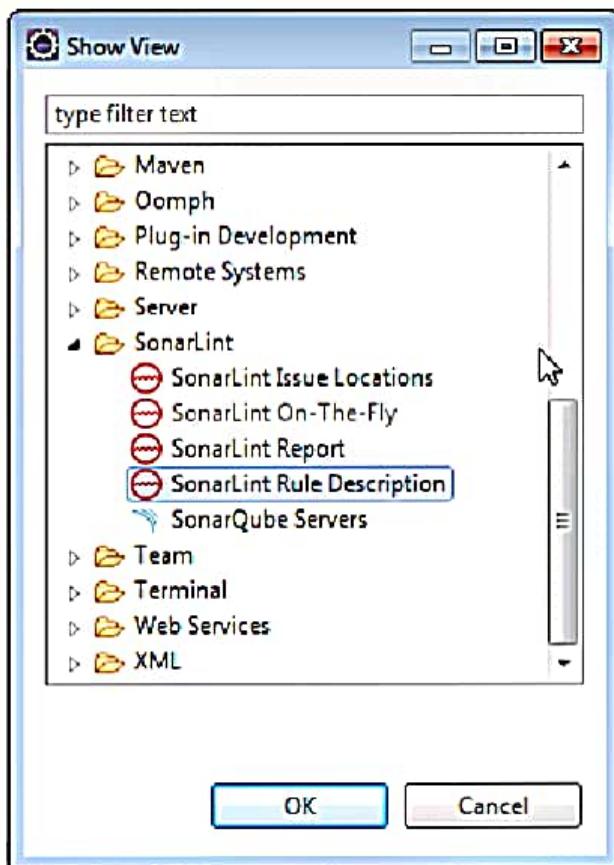


A new tab opens below which shows the issues found in the current java file selected.

Date	Description	Resource
	Either log or rethrow this exception.	calculator.java
	Remove this useless assignment to local variable "C".	calculator.java

Issues reported "on the fly" by SonarLint on files you have recently opened/edited

Step 2: Navigate to Windows>Show view>other. Here select SonarLint>SonarLint Rule Description.



A new empty SonarLint Rule Description tab opens. Now select one of the issue in the SonarLint On-the-fly tab and switch to SonarLint Rule Description tab. This suggests the necessary changes to be made in the code to retain the quality of the code.

Exception handlers should preserve the original exceptions (squid:S1166)

Bug Major

When handing a caught exception, the original exception's message and stack trace should be logged or passed forward.

Noncompliant Code Example

Estimated time: 20 mins

Summary: You have learnt the static code analysis using SonarLint in this exercise

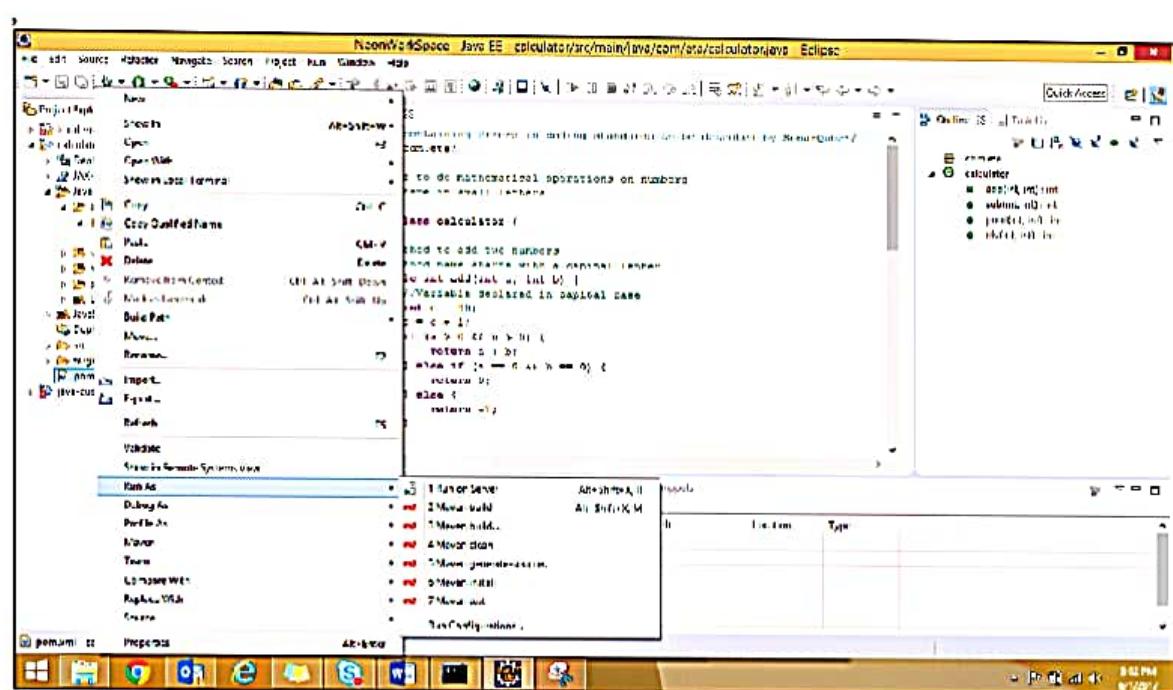
3. Binding SonarQube rules to SonarLint

Objective: Customize the rules used by SonarLint through SonarQube web server interface.

Requirements: SonarQube 5.6 and above

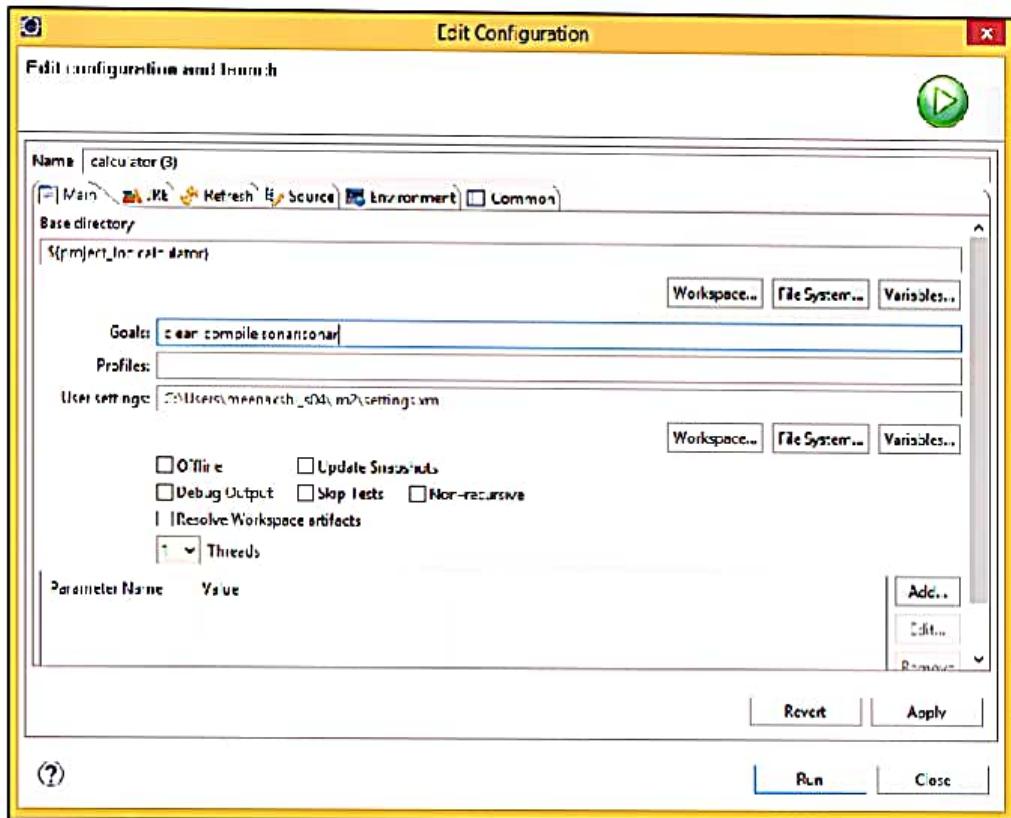
Step 1: Run the StartSonar.bat  file as admin.

Step 2: In the project imported in the previous exercise, run the pom.xml as Maven build.



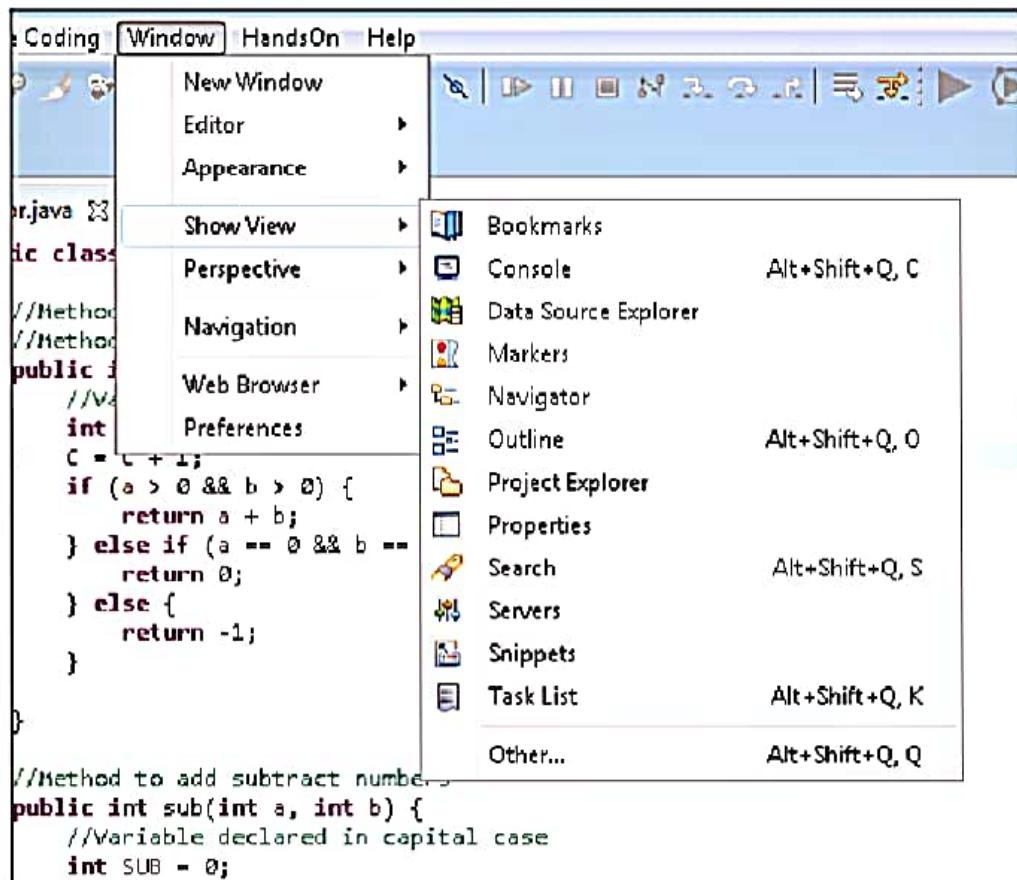
In the Goals tab, use clean, compile and sonar:sonar and execute the pom.xml file

Assignments for DevOps Continuous Integration and Continuous Delivery

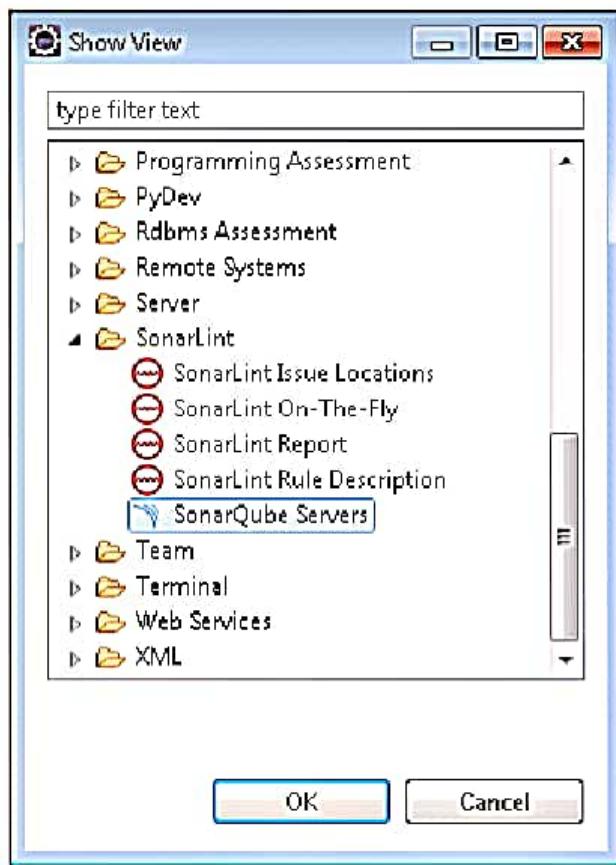


Once the build is successful, we need to bind the current java project with the SonarQube project.

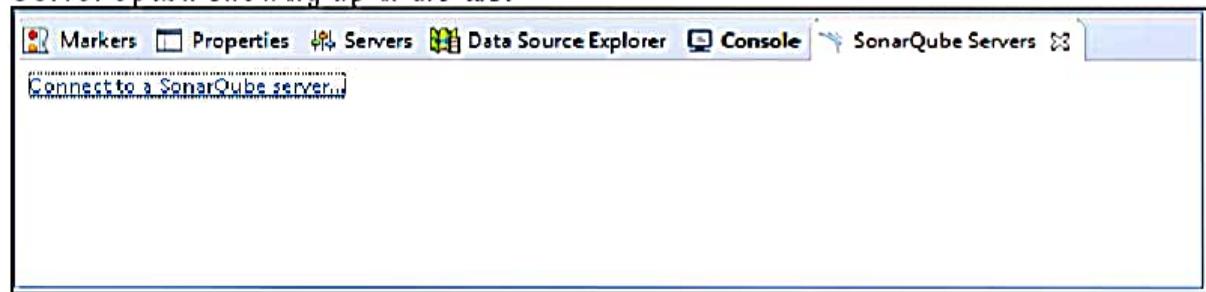
Step 3: Go to Window->Show View->Other.



Select SonarLint->SonarQube Server.

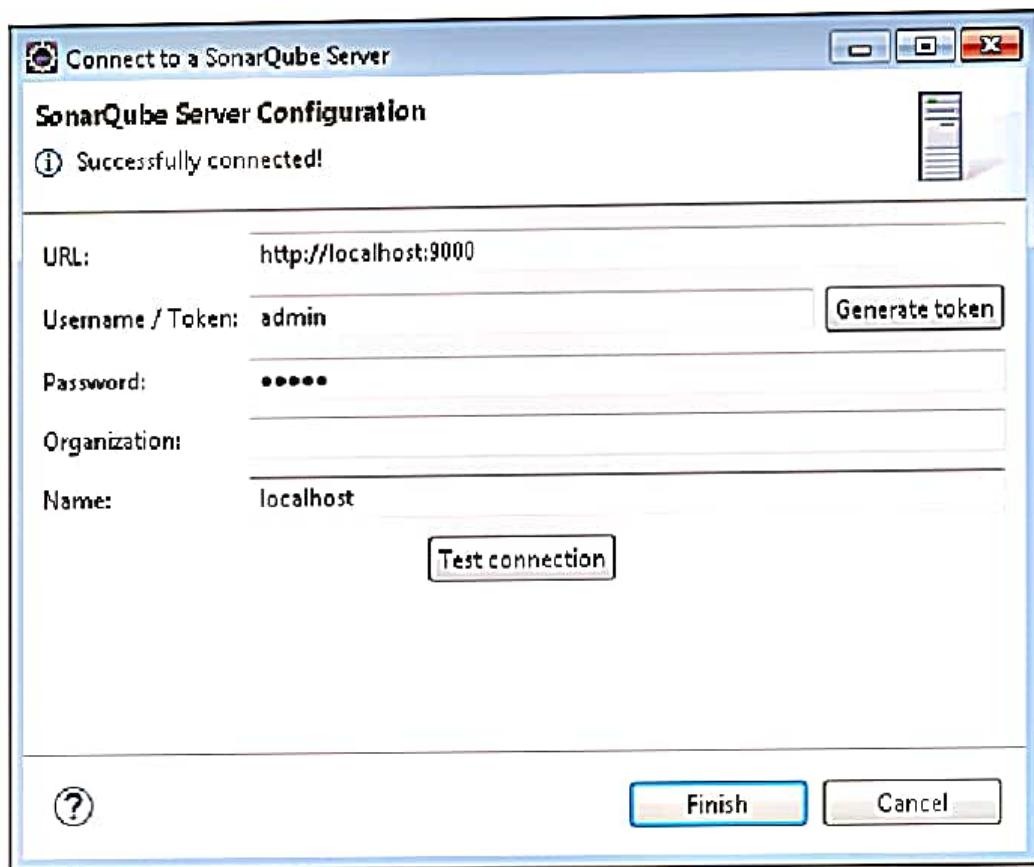


This opens a SonarQube Servers tab below. Click on the Connect to a SonarQube Server option showing up in the tab.

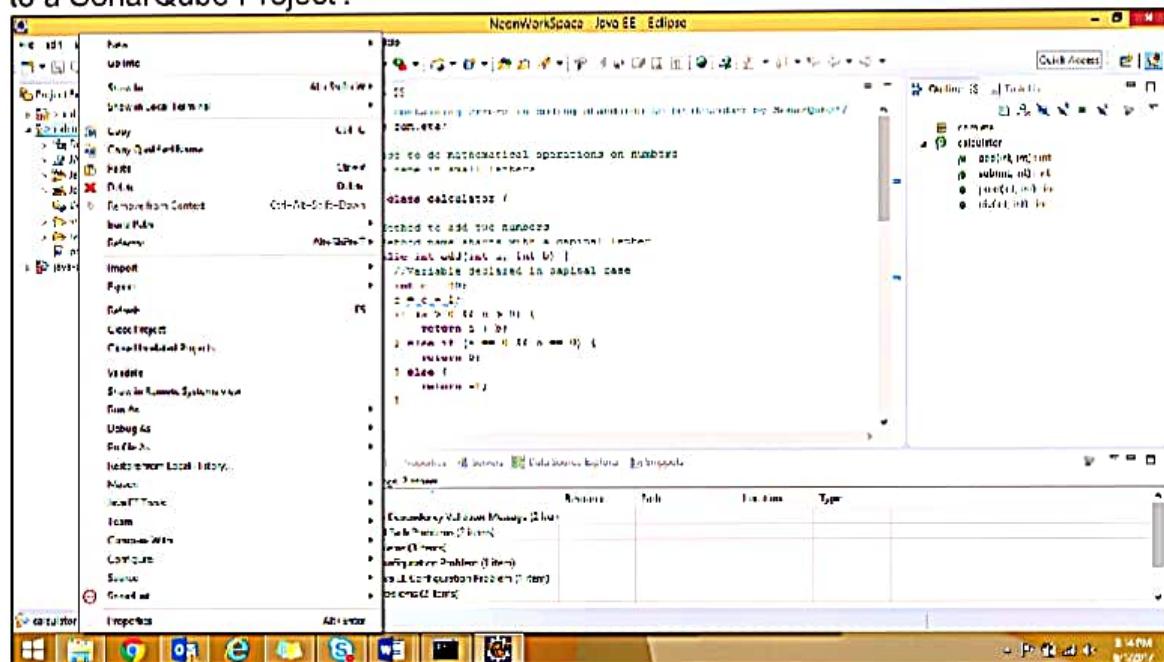


Step 4: Put the following details Username:admin Password:password as shown in the image given below and then click on TestConnection button.

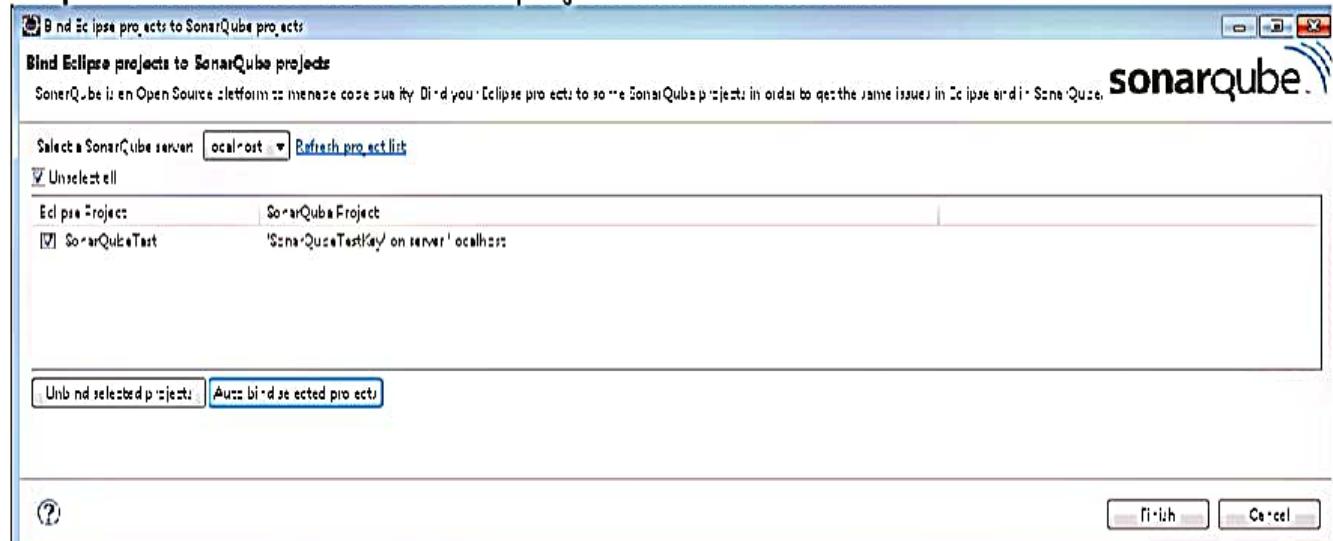
Assignments for DevOps Continuous Integration and Continuous Delivery



Step 5: Go back to Eclipse, right click on the project under SonarLint click on 'Bind to a SonarQube Project'.



Step 6: Click on Auto bind selected projects and click on Finish.



Once you click on Auto bind, you will see 'SonarQubeTestKey/ on server localhost' under the title **SonarQubeProject**

Step 7: Now open a browser and open <http://localhost:9000> to view the project on the SonarQube server.