Homework 1

Team members:
Adarsh Hegde ahegde5@uic.edu
Ashwani Khemani akhema2@uic.edu
Srinath Karimpuzha skarim25@uic.edu

OBJECTIVE

The objective of the project is to create a Dev-Ops pipeline for automatically building, testing and analyzing software applications.

I.   **Instruction on how to create the Environment Required for the project**

Operating System: The entire project has been simulated on both Linux and Windows environment.

 Installation of Software tools required for creating a Dev-Ops pipeline
1) Git:
  Follow the below link to install Git
  https://www.atlassian.com/git/tutorials/install-git

2) GitLab Server:
  For Ubuntu, Gitlab Server was installed natively
  https://about.gitlab.com/installation/#ubuntu

  Note: Kindly set the EXTERNAL_URL and port number according to user's preference. In our setup we have set the external URL as http://localhost.We will use the same URL to access the Gitlab Server. During the first login into Gitlab kindly set the password for the root user.

  For additional Configuration kindly refer
  https://docs.gitlab.com/omnibus/README.html#installation-and-configuration-using-omnibus-package
  Kindly reconfigure and restart gitlab after changes in the configuration. Reconfigure and restart commands are as follows.

  sudo gitlab-ctl reconfigure
  sudo gitlab-ctl restart

  For Windows, Gitlab is not supported, hence Docker was used to run Gitlab inside a Docker Container
  https://docs.gitlab.com/ee/install/docker.html

Gitlab: git pull gitlab/gitlab-ce
docker run -d --name my-gitlab -it -p 80:80 -p 43:43 -p 22:22 gitlab/gitlab-ce

Generating an access token to access Gitlab API
a. Open Gitlab using Gitlab URL (in our case http://localhost).
b. Go to User Setting → Access Tokens
c. Generate access token and ensure you select scope namely api, read_user and sudo.
d. **Kindly note down the access token for future reference**

3) Jenkins
For windows kindly use the below link
https://jenkins.io/doc/book/installing/#windows

For Linux environment kindly use the below link
https://jenkins.io/doc/book/installing/#debian-ubuntu

Kindly ensure that there is no port conflict between Jenkins and Gitlab server. By default both use port 8080. So kindly change the port number in Jenkins config(/etc/default/jenkins) file as explained in the documentation above. Also restart the jenkins server.
Note: If Jenkins doesn't have entry for HTTP_PORT, add following line:
HTTP_PORT=8081

4) Oracle JDK
https://medium.com/coderscorner/installing-oracle-java-8-in-ubuntu-16-10-845507b13343

5) Install Maven

II. Configuring the environment.

In this step we proceed with Gitlab and Jenkins integration.
To integrate Gitlab with Jenkins we need to configure two credentials in Jenkins credentials store.
Steps to create credentials:
1. Open the Jenkins URL and follow suggested steps on the page.
2. Installation of Plugins
        Now we install the following Plugins from:
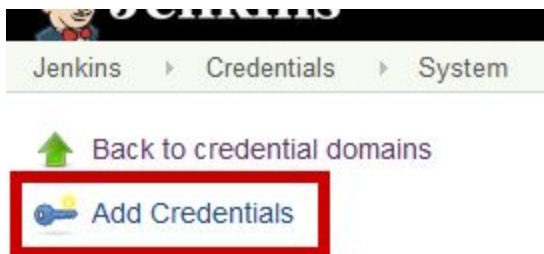   Manage Credentials → Manage Plugins option.
        a. Gitlab Plugin
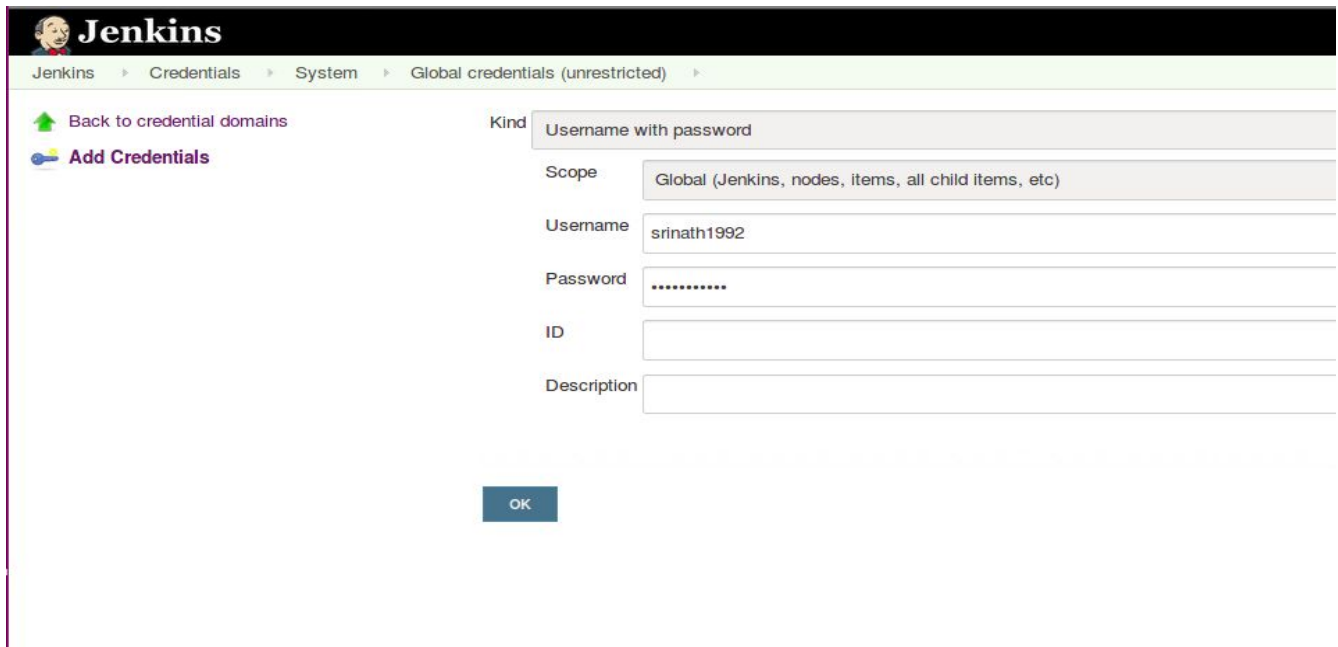   b. Jacoco Plugin
3. Go to Credentials tab

4. Go to Systems Option.

5. Go to Add Credentials Tab.

6. The first credential is of Kind Username and Password. ALso provide a unique ID.
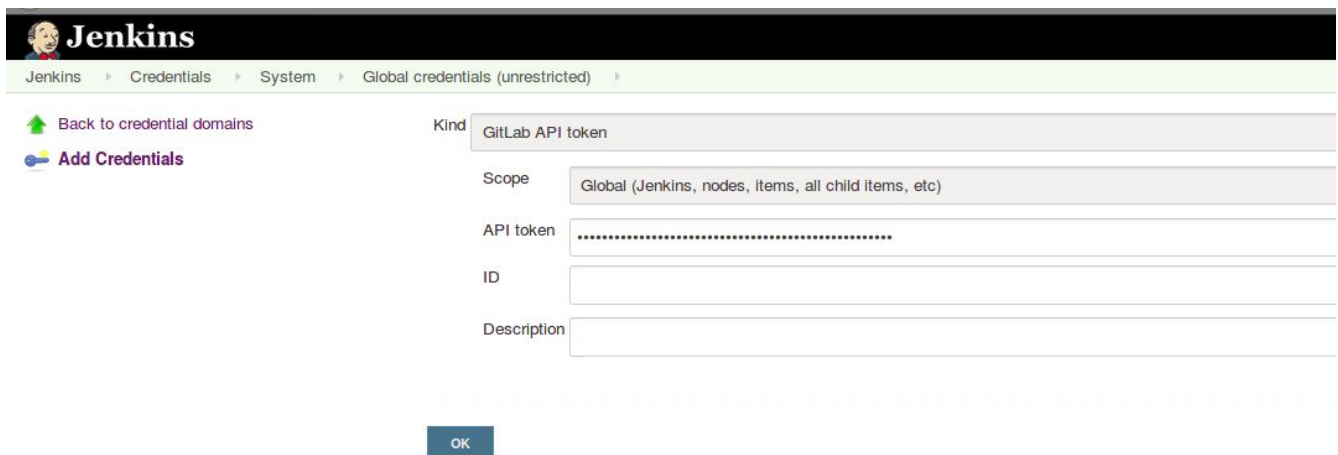
7. The second credential is of type Gitlab API Token. Here we make use of the access token which we generated during the setup of Gitlab.(Done in step 2 d.)

8. To verify the connection of Jenkins with Gitlab go to
Manage Jenkins → Configure System and set the following Gitlab credentials.

Kindly choose Gitlab API token from drop down, enter Gitlab URL and test the connection.

Gitlab

Enable authentication for '/project' end-point	☐
GitLab connections	

Connection name: organization gitlab

A name for the connection

Gitlab host URL: http://localhost:80

The complete URL to the Gitlab server (i.e. http://gitlab.org)

Credentials: GitLab API token (Jenkins Server X API to Gitlab) ▾ 🔑 Add ▾

API Token for accessing Gitlab

Advanced...

Test Connection

Delete

Add

9. Set up the global configurations for Jenkins:
This will include setting up the path for JDK, Git, Maven or gradle.To do this configuration Go to Manage Jenkins-->Global Tool Configuration and set up the required environment paths.

JDK

JDK installations

JDK Name: jdk_home

JAVA_HOME: /home/askheman/Downloads/jdk1.8.0_161

☐ Install automatically

Delete JDK

Add JDK

List of JDK installations on this system

Git

Git installations: Add Git ▾

Gradle

Gradle installations...

Ant

Ant installations...

Maven

Maven installations

Maven Name: mvn_home

MAVEN_HOME: /usr/share/maven

☐ Install automatically

Delete Maven

Add Maven

List of Maven installations on this system

Docker

III. Running the Code

We will now do the necessary changes for the input parameters for our code.
Kindly setup any IDE like IntelliJ or Eclipse in order to work on provided code .
We have to set up JDK for the IDE and Gradle for the project.
We will make some changes in the properties file present in the project root directory .

Open the properties file in the IDE and change the following attributes as per your system :

```
# Code destination on file system
codeDestination= D://gitrepo/

# Github username
githubUserName=<Enter your Github username>
# Github password
githubUserPassword= <Enter your password>



# Limit of number of projects pulled from github
projectLimit=20

# Language type of projects
projectLanguage=Java

# Lower limit of project size
projectSizeLowerLimit=100

# Type of projects - maven/ant/gradle etc
projectType=maven

# Gitlab host url
gitlabHost=http://localhost

# Gitlab username
gitlabUserName=root

# Gitlab user password
gitlabUserPassword=<Enter your password>

# Gitlab access token
gitlabAccessToken=<Enter your access token (Refer 2 d) )>

# Jenkins host Url
jenkinsHost=http://localhost:8080/
```

```
# Jenkins user name
jenkinsUserName=<Enter your Jenkins username>


# Jenkins User password
jenkinsUserPassword=<Enter your Jenkins password>


# Jenkins Job Url - Will be used to connect from Gitlab to Jenkins
# This will be different from Jenkins host url if you are working in Docker
jenkinsJobUrl=http://host_ip:port/project/


# Location of Jenkins config file
#Place the given config.xml File available in the project root directory at any location in
your system and provide the necessary path here.
jenkinsConfigFileLocation=<Path of file>




#Jenkins home directory
#Location of where Jenkins is installed where workspaces and jobs are created
jenkinsHome = <Path of Jenkins Home>
```

After doing the changes in the properties file , we will now the run the application .
The applications will fetch a given number of java - maven based applications using the configuration given in the properties file .
1. Application will create the projects in the Gitlab server
2. It will create the respective Jenkins job for the project
3. Then it adds the webhook for the project in Gitlab.
4. The code corresponding for the projects is then pushed in the Gitlab server which will trigger the build in the Jenkins server ,generate Reports and dependency matrix with Understand and Jacoco test reports .

If any of the above functionality does not work correctly please ensure that the webhook was correctly setup . The webhook can be tested by going into the corresponding project Settings -- Integrations tab.

If the webhook is correctly configured , we get the below message on the UI .

If Webhook fails when the given error :
Jenkins Anonymous user does not have job/build permission , kindly uncheck the Enable authentication for '/project' endpoint check box as below
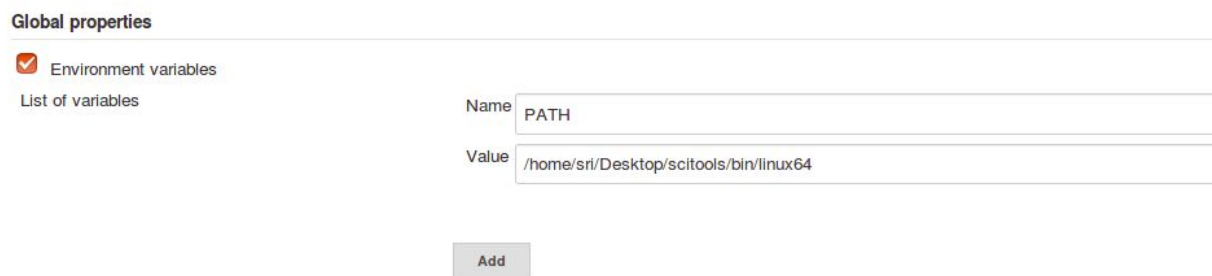Manage Jenkins → Configure System and set the following Gitlab credentials.

If the Webhook fails with the given error :
No valid crumb was included in the request , please check the webhook URL and ensure the URL is in the correct format : http://jenkins_host_ip:port/project/project_name

IV. Configuring Scitools Understand :

1. Kindly download Understand API from scitools website https://scitools.com/non-commercial-license/
2. Keep a note of the directory where the Understand API is downloaded and extracted in your local system.
3. Kindly traverse inside the extracted scitools folder and go to scitools/bin/<Your System> / and check if "und" executable is present.
4. Capture the path/location of the directory of und executable.
5. Now set the path of und in Jenkins .Go to Jenkins Manage Jenkins → Configure System
6. Go to Global Properties and set PATH in name and the path where und executable is present in Value.

Global properties

☑ Environment variables
List of variables

Name	PATH
Value	/home/sri/Desktop/scitools/bin/linux64

Add

The und commands for generating report and dependency matrix are passed through the config.xml and executed in the shell of each job.

UND Script:-
und create -db Report.udb -languages java
und add -db Report.udb \src
und analyze -all Report.udb
und export -dependencies class matrix Dependency Matrix.csv Report.udb
und report Report.udb
und metrics Report.udb

- The Reports, Dependency Matrix and metrics for each job are generated in the workspace folder of jenkins home.
- Reports are generated in Report.txt file
- Dependency Matrix in generated Dependency Matrix.csv

V. Configuration the Jacoco Reports for Test Coverage :

We made use of the Jenkins Jacoco Plugin to generate the test coverage reports .

Recording coverage for builds :
Get coverage data as part of your build

First we need to get coverage calculated as part of our build/tests . We need at least one or more *.exec file available after tests are executed. This is done by doing the necessary changes in the Maven pom.xml or Ant build.xml file.

It might be the case that the projects cloned from the github do not have necessary changes to extract the Jacoco Test reports . Please do the below change in the Pom.xml file for doing the configuration required for Jacoco reports .

```
<plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.0</version>
        <configuration>
                <destFile>${basedir}/target/coverage-reports/jacoco-unit.exec</destFile>
                <dataFile>${basedir}/target/coverage-reports/jacoco-unit.exec</dataFile>
        </configuration>
        <executions>
                <execution>
                        <id>jacoco-initialize</id>
                        <goals>
                        <goal>prepare-agent</goal>
                        </goals>
                </execution>
                <execution>
                        <id>jacoco-site</id>
                        <phase>package</phase><goals>
                        <goal>report</goal></goals>
                        </execution>
                </executions>
</plugin>

<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration><source>1.5</source>
        <target>1.5</target>
        </configuration>
</plugin>
```

Set up coverage retrieval and publishing

In order to get the coverage data published to Jenkins, we need to add a JaCoCo publisher and configure it so it will find all the necessary information. Basically we specify where the *.exec files are, where compiled code can be found and where the corresponding source code is located after the build is finished to let the plugin gather all necessary pieces of information.

This is done by adding a post build step in the Jenkins job . This changes for this has already been done in the provided Jenkins config.xml .

Please refer the below documentation for more details :
https://wiki.jenkins.io/display/JENKINS/JaCoCo+Plugin

We ran the Jenkins - Jacoco plugin to extract the test coverage reports for a open source github java project - jodatime3 . The results of the test coverage have been shown below :
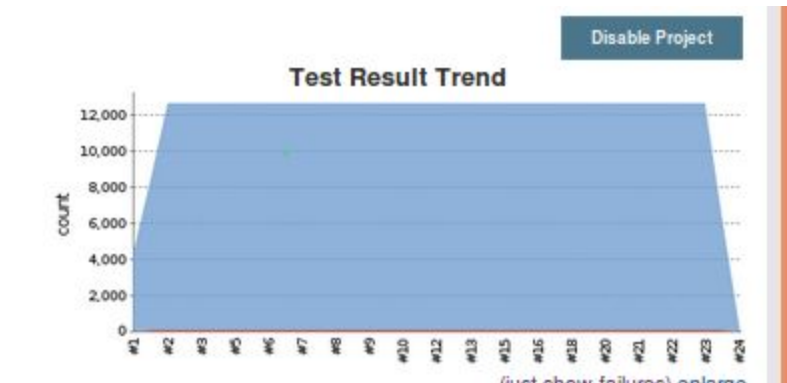
Test Result

1 failures (±0)

12,671 tests (±0)
Took 17 sec.
add description

All Failed Tests

Test Name	Duration	Age
junit.framework.TestSuite$1.warning	20 ms	2

All Tests

Package	Duration	Fail (diff)	Skip (diff)	Pass (diff)	Total (diff)
junit.framework	20 ms	1	0	0	1
org.joda.time	2.4 sec	0	0	9018	9018
org.joda.time.chrono	12 sec	0	0	516	516
org.joda.time.chrono.gj	1.3 sec	0	0	4	4
org.joda.time.convert	76 ms	0	0	561	561
org.joda.time.field	65 ms	0	0	837	837
org.joda.time.format	0.72 sec	0	0	1650	1650
org.joda.time.tz	0.16 sec	0	0	84	84

Test Result Trend

Disable Project

Jacoco - Overall Coverage Summary

INSTRUCTION	90%	
BRANCH	82%	
COMPLEXITY	82%	
LINE	91%	
METHOD	91%	
CLASS	99%	

All the corresponding test reports for the project can be find in the workspace folder in the jenkins home directory .

VI. Patch using GitHub Developer API

Using GitHub Develop API command we are able to get the latest two changes and the lines of code that were changed in those previous commits.*This metainformation is useful for Github Developers and testers for creating a patch for the next changes to be added or for testing the patch.*The patch is stored in Patch_Projectname.txt in the directory where the project has been cloned from GitHub.

Patch for a simple Palindrome Application generated programmatically where earlier commit has changed directory structure and boolean variable from true to false.

```
diff --git a/primitives/src/test/java/CheckIfIntegerIsPalindromeTest.java b/primitives/src/test/java/CheckIfIntegerIsPalindromeTest.java
index 4b3b35d..e4cd884 100644
--- a/primitives/src/test/java/CheckIfIntegerIsPalindromeTest.java
+++ b/primitives/src/test/java/CheckIfIntegerIsPalindromeTest.java
@@ -25,7 +25,7 @@

    @Test
    public void isPalindrome3() {
-        expected = true;
+        expected = false;
        input = -1;

        test(expected, input);
```

VII. Simulation and Testing

How to run the application?
Please resolve all your dependencies before running the application and make sure all the configurations are updated in devops.properties.

Finally, execute the Main.java file.

How to build the application?
`gradle build`

Running the unit tests
Test classes for all the possible scenarios are implemented. JUnit has been used for performing Unit Testing.

How to run the test suit?
`gradle test`

Result:

Test Summary

8	0	0	39.180s
tests	failures	ignored	duration

100%
successful

Packages   Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.uic.atse.service	7	0	0	39.180s	100%
com.uic.atse.utility	1	0	0	0s	100%

Known Issues:

None