CS 540 - Homework 1(DevOps Pipeline Simulation)

Team members: Pratik Kshirsagar(pkshir2@uic.edu) and Sai Phaltankar(sphalt2@uic.edu)

Overview:
Our task was to simulate a DevOps pipeline with multiple components such as Jenkins CI, our own GitLab server and SciTools Understand, a static code analyzer.

What we did:

1. We installed Docker(Community Edition) so as to run a GitLab server as a container on our own local machines.
2. We installed and configured Jenkins on our system, locally.
3. We set up a GitLab server and then connected our Jenkins instance to our GitLab server.
4. The next step was importing open source repositories from GitHub and pushing them onto our GitLab server.
5. Once we cloned the repositories onto our GitLab server, we create Jenkins jobs that trigger a CI pipeline when there is a push event in any of the GitLab repositories.
6. For this we needed to set up Webhooks between GitLab repositories and respective Jenkins jobs.
7. We then used a static code analysis tool called Understand to generate Dependency graphs and other Analysis reports.
8. We also used the "git diff" command to find out certain files that may required to be retested due to their newly modified state.

A detailed explanation of the above steps is given below.

Generating SSH Keys: SSH keys are required for secure communication between Jenkins, GitLab server and our local machine which hosts these services. It is easy to create SSH keys as follows:

```
ssh-keygen -t rsa -C “your_email@example.com"
```

You can copy the generated ssh key onto clipboard as follows:
`pbcopy < ~/.ssh/id_rsa.pub` —> for copying the public part
`pbcopy < ~/.ssh/id_rsa` —> copying the private part

Installing Docker: We used Docker to run GitLab server in a container. To install docker, we followed these steps (We used a Mac). It was a relatively simple process, a regular application installation through a wizard. The installer can be found here: https://store.docker.com/editions/community/docker-ce-desktop-mac

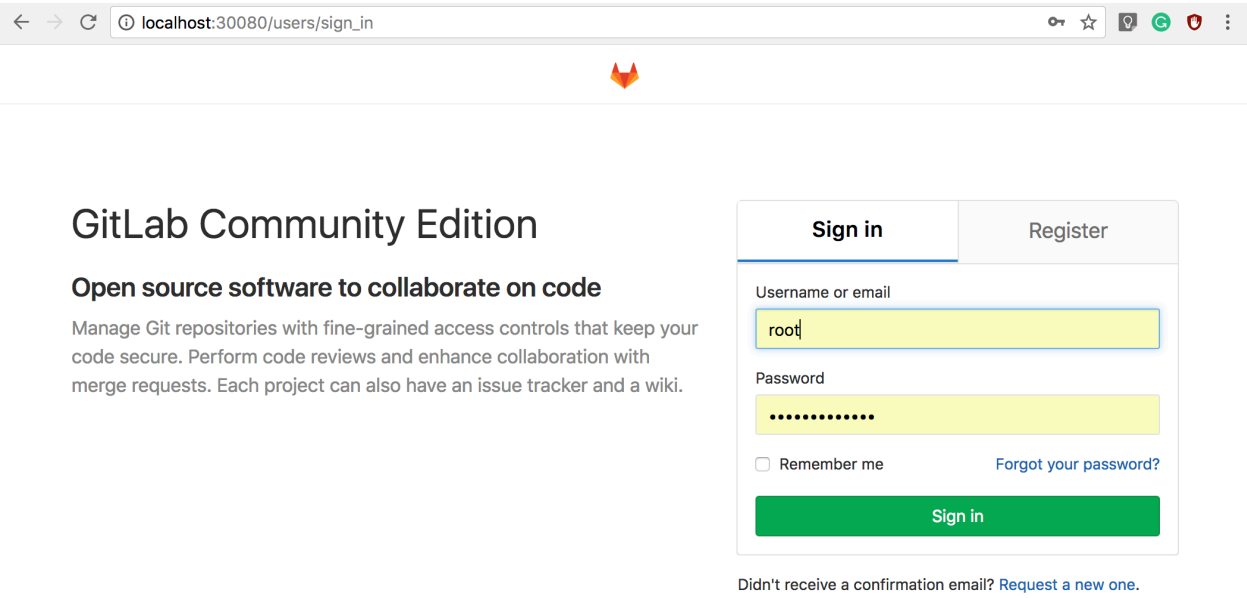
Installing GitLab: We ran GitLab CE as a container. So to get the server up and running, we had to execute the following script on the terminal. I have used my /Users/.../CS540/gitlab directory to create volumes for the container. This will help the container store data on my machine. Please change the path as per convenience. We have exposed various ports such as HTTP, HTTPS and SSH. The external URL points to the homepage of GitLab.

```
sudo docker run --detach --name gitlab \
     --hostname gitlab.example.com \
     --restart always \
     --volume /Users/pratikkshirsagar/Desktop/CS540/gitlab/config:/
etc/gitlab \
    --volume /Users/pratikkshirsagar/Desktop/CS540/gitlab/logs:/var/
log/gitlab \
    --volume /Users/pratikkshirsagar/Desktop/CS540/gitlab/data:/var/
opt/gitlab \
     --publish 30080:30080 \
     --publish 20080:80 \
     --publish 30022:22 \
     --publish 32768:443 \
     --env GITLAB_OMNIBUS_CONFIG="external_url 'http://
gitlab.example.com:30080';
gitlab_rails['gitlab_shell_ssh_port']=30022;" \
     gitlab/gitlab-ce:latest
```

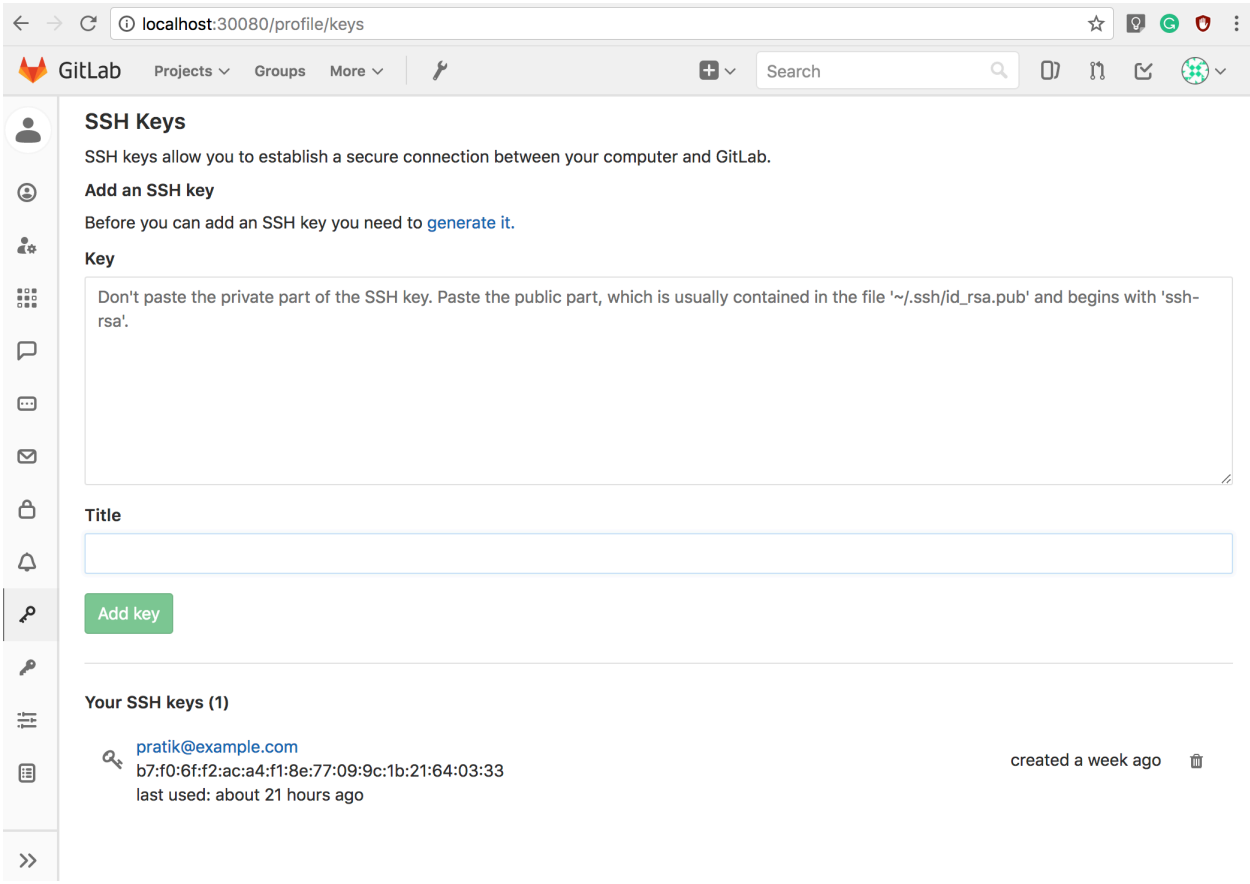
Configuring GitLab: Configuring GitLab was done entirely through the GUI. Once the container is up and running, which may take a few minutes to spin up, we can access GitLab at “http://localhost:30080" as we’ve set that as our external URL. We have to set up a “root” administrator account which has access to the entire server. We navigate to the “settings” panel from the drop down menu from the user icon at the top right corner. Next,

we set up our SSH key and a Private Access Token which can be used to authenticate API calls later used with Jenkins. Please refer the attached images.

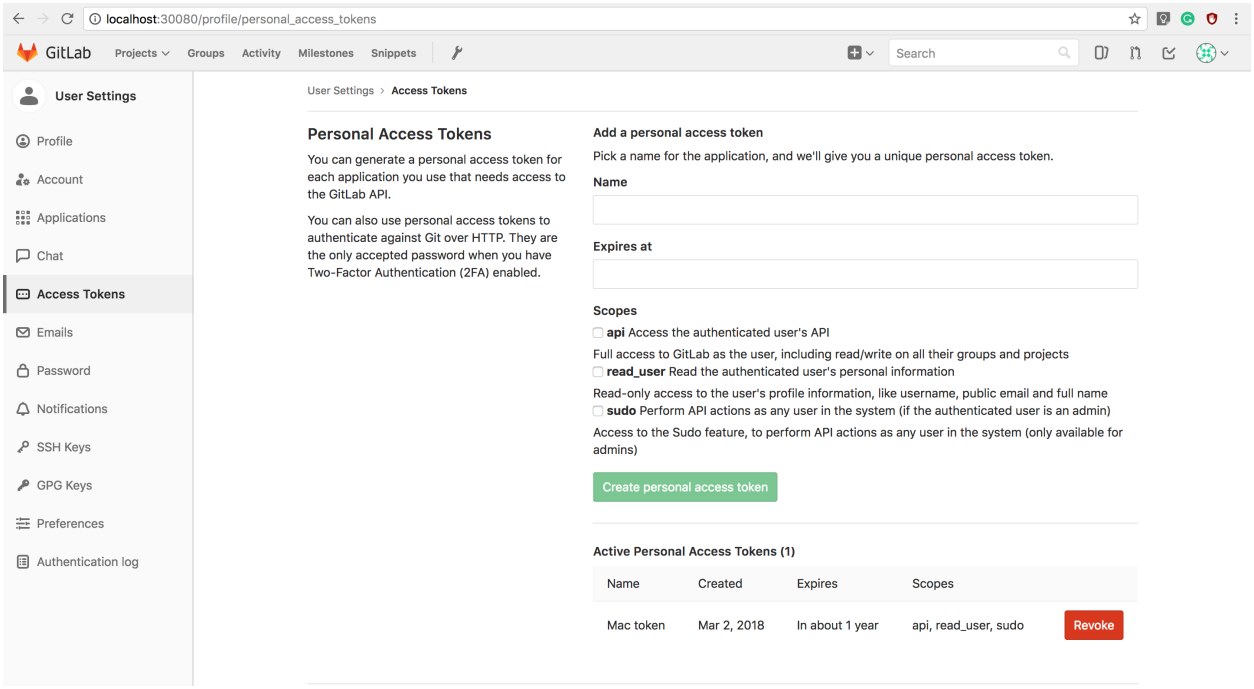
Signing in as the root user for administrator-level access.



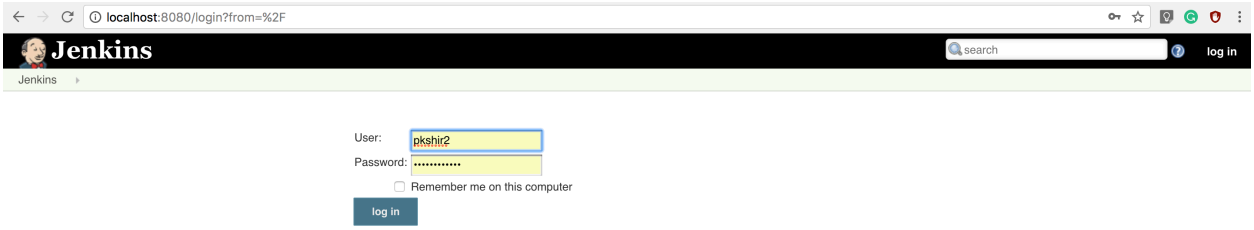
Adding SSH keys.



Creating Personal Access Tokens. (Please save them somewhere)



Installing Jenkins: We installed Jenkins natively on our machine. The installation process was again simple. The wizard will automatically install Jenkins onto our system and then we have to set up an administrator account, initially with the hidden password, which we are prompted to change later on. The GUI is self-explanatory. By default, the Jenkins instance can be accessed at "http://localhost:8080".

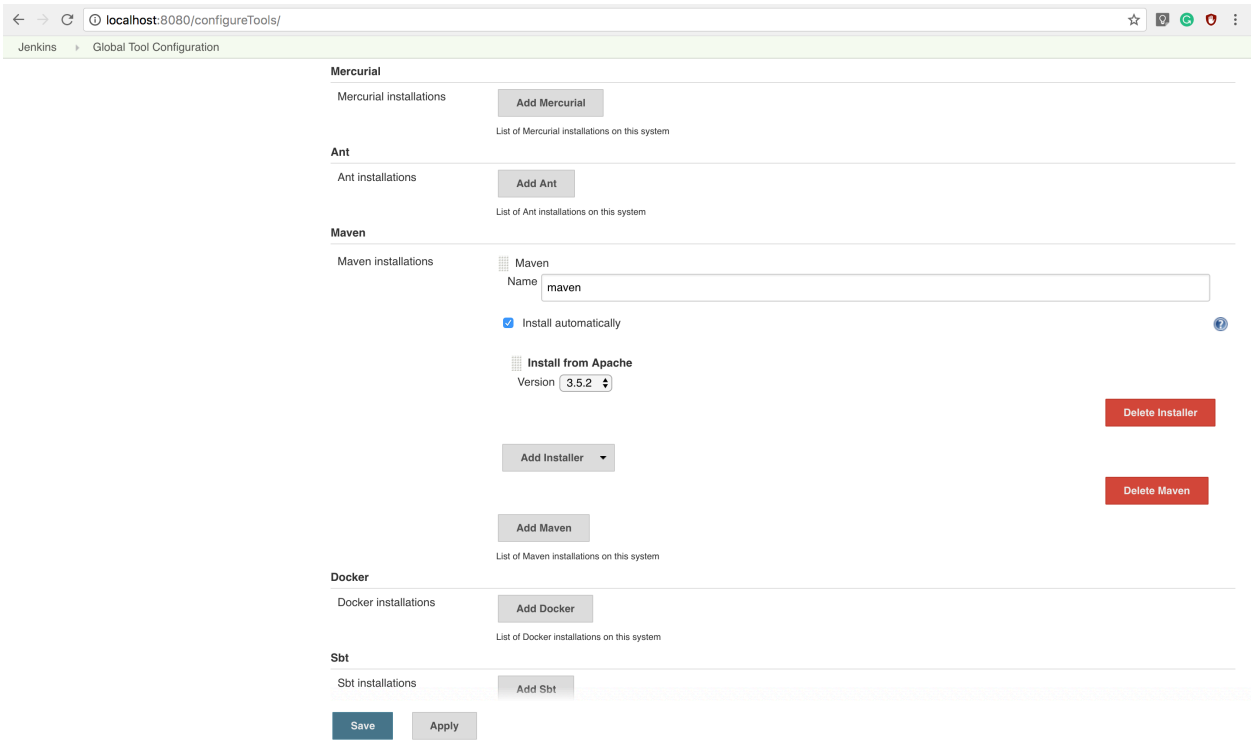


On your first run, the Jenkins instance will ask you for the temporary password. You can copy it using the command:

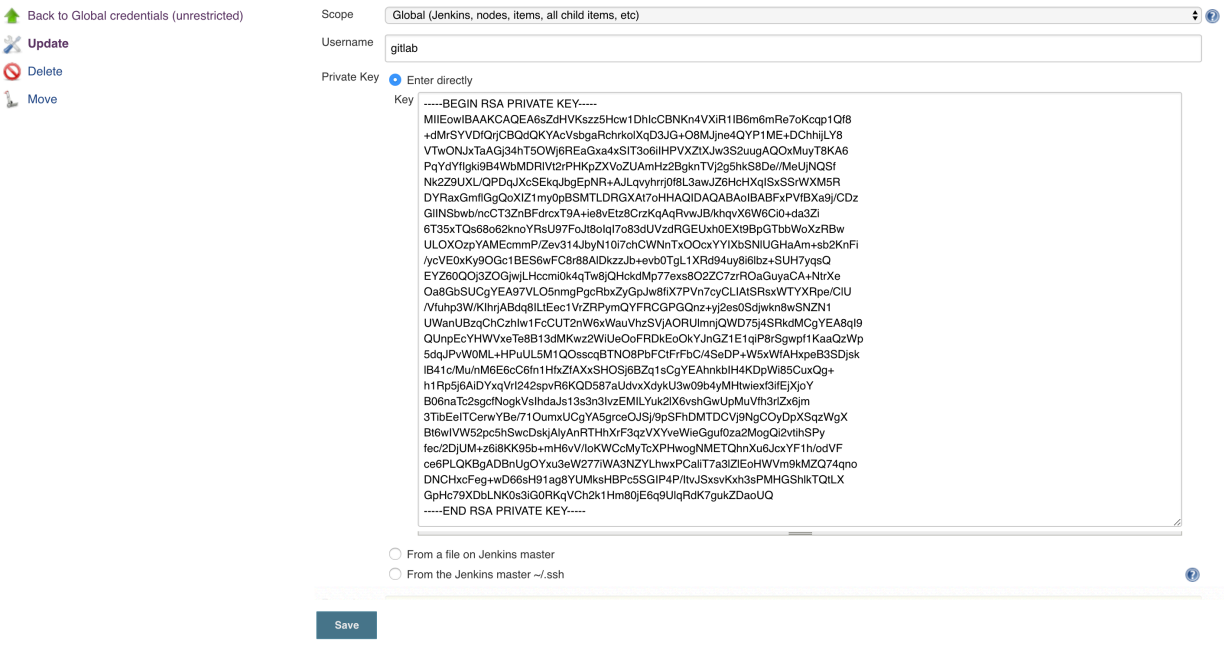
```
sudo cat /Users/Shared/Jenkins/Home/secrets/initialAdminPassword
```

Follow the default installation instructions per the screen and allow the default plugins to be installed. We will later install all the plugins we need to run our Jenkins jobs.

Configuring Jenkins: We need to install the following plugins: Jacoco, Job DSL, XML Job to Job DSL, All the Git plugins, All the GitLab plugins, All the Maven plugins. We then set up a maven installation for Jenkins as we will be using maven builds as part of our pipeline. This was done through Configure Jenkins -> Global Tool Config.

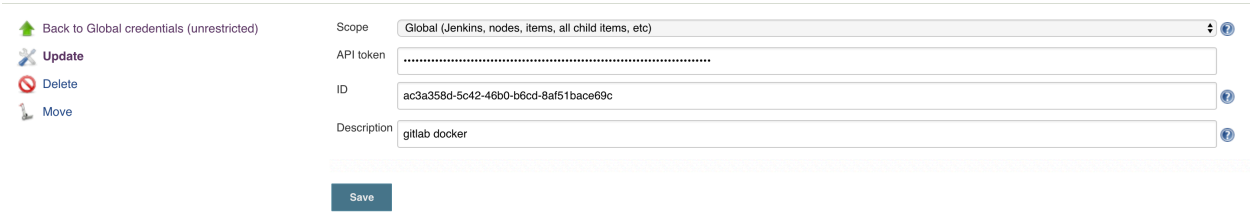


We need to create Jenkins global credentials in order to establish a connection between Jenkins and GitLab. This can be done through Credentials -> System -> Global -> Add credentials. We add two kinds of credentials: SSH key and GitLab Personal Access Token. Both of which were generated earlier.

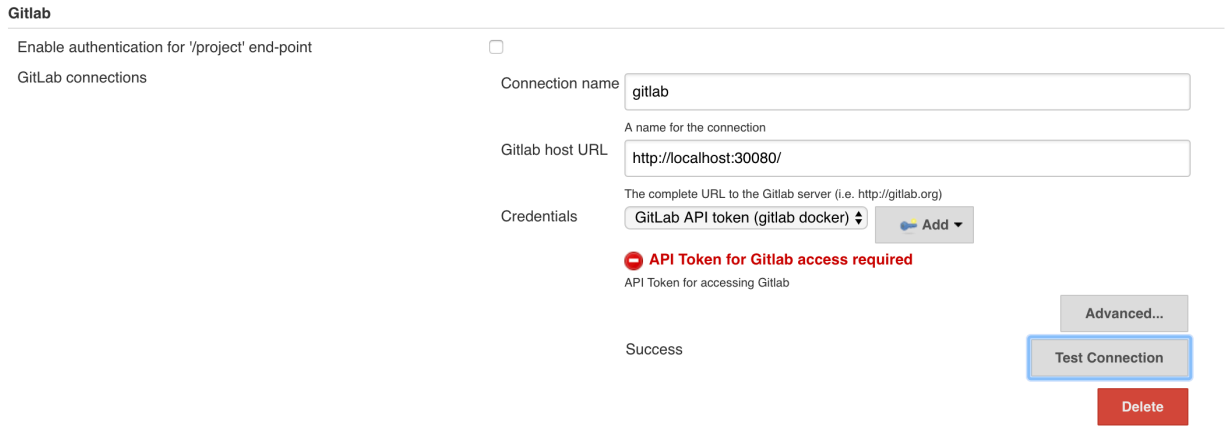


Adding SSH Key.

Adding GitLab API token.



Integrating Jenkins with GitLab: We go to Manage Jenkins -> Configure System and add the GitLab host and credentials. We then test the connection, if successful that means our integration is ready to use. Make sure to select the right credentials and host URL.



Integrating Jenkins with GitLab.

Installing Understand: Install SciTools Understand. The professor has given us details of obtaining a educational license. With that, installing the application through a wizard is easy. Once installed, please add path of the und executable to the bash settings. (Export the Path variable.) We will be using the python API hence we need to add another path variable.
PATH=/Applications/Understand.app/Contents/MacOS/Python:$PATH
PATH=/Applications/Understand.app/Contents/MacOS:$PATH

The following part assumes that you have python3 installed on your system, along with the pip package installer. You can install python3 from here and pip from here.

Installing the dependencies: execute the following command from the root of the project directory, which contains the requirements.txt file.

```
sudo pip3 install -r requirements.txt
```

Cloning Repositories from GitHub to GitLab: We have used GitHub's REST API to filter out repositories that qualify two criteria: their language is "Java" and their build system is "Maven".

The query is as follows:

```
r = requests.get('https://api.github.com/search/repositories?q=language:java+topic:maven', auth = ("username", "password"))
```

We are essentially using the requests package in python that let's us deal with HTTP requests. This request gives us a dump of repositories in JSON file. By parsing through the JSON file, we are able to clone each repo onto GitLab with the help of GitLab's REST API.

The request is as follows:

```
g = requests.post('http://localhost:30080/api/v4/projects', data = {'name':repo_name,'import_url':repo_url}, headers = {'Private-Token':'PRIVATE-TOKEN'})
```

Note that this POST request is accessing our GitLab host and creating a new project for every such request. We are using a single loop through the JSON file and cloning everything in one go.

Creating Jenkins jobs: We create a Jenkins job which is essentially the pipeline that will be executed when the job runs. We first create a dummy job through the UI and then extract it's config.xml. We then modify the XML for every job and create a job for every GitLab repository through a loop. We are using the "python-jenkins" and "python-gitlab" packages in order to do so.

Structure of each Jenkins job: We have to specify the source repository URL, the credentials, the build triggers, the build step and the post build actions. For our Jenkins jobs, the configuration is as follows:

Source Management : Specify the clone URL of GitLab repository.
Credentials : SSH Key.
Build Trigger : Build when a change is pushed to GitLab (Gitlab plugin comes handy.)
Build Step: Maven build with tasks; clean compile test.
Post Build Actions : Record Jacoco Coverage Report, Report Build Status to GitLab.

We feed this configuration into a job config.xml (XML to DSL plugin comes in handy.) Then using python-jenkins we can iteratively create Jenkins job for each GitLab project. Please refer to ‘output.xml’ as a sample.

Setting up Webhooks for projects -> jobs: With the help of web hooks, every time we push a change into GitLab, the corresponding Jenkins job is triggered. We are creating Webhooks with the python-gitlab package.

The following snippet is an example of creating Jenkins jobs and setting up Webhooks for a project.

```
server = jenkins.Jenkins('http://localhost:8080/', username='pkshir2',
password='jenkinsrocks') # Pass Jenkins host URL and credentials.
user = server.get_whoami()
print('Hello %s from Jenkins' % (user['fullName']))

gl              =              gitlab.Gitlab('http://localhost:30080',
private_token='9SsFv3LD6ijhGznMSjuK') # Pass the gitlab host URL and
generated private API token (from gitlab.)
gl.auth()

hooks = project.hooks.create({'url': 'projectURL', 'push_events': 1})
```

Jacoco Coverage Report: If the project contains a .exec file for Jacoco to execute, then the report contains the respective metrics or else it’s just empty report without any errors.

Maven Build: We have included the following tasks; clean, test, compile.

Generating Understand Reports: We use understand to generate Dependency Graphs and Pie Charts containing post analysis code metrics. We achieve this by using a combination of Understand’s command line (und) and Understand’s python API. Please refer to ‘understand_report.py’ for specifics. Since the API is read-only, we used ‘und’ to create an Understand DB and then the API to add projects and perform analysis.

Generating Test Recommendations: Using ‘git diff’ command we generate a text file called ‘RetestTheseFiles.txt’ that tells you which files have been modified in the last 3 commits and need to be retested due to their newly modified state. Refer ‘clone.py’ for specifics.

Acceptance Tests: Written some acceptance tests that check whether the HTTP requests that every component is making is actually working correctly as intended.

Status of GitLab container(healthy)

```
git
   af51570..e0cd637  master -> master
[Pratiks-MBP:Homework pratikkshirsagar$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND             CREATED
      STATUS                            PORTS
                                                                 NAMES
8572bd1c7074        gitlab/gitlab-ce:latest        "/assets/wrapper"   2 weeks ago
      Up 9 seconds (health: starting)   0.0.0.0:30080->30080/tcp, 0.0.0.0:30022-
>22/tcp, 0.0.0.0:20080->80/tcp, 0.0.0.0:32768->443/tcp   gitlab
[Pratiks-MBP:Homework pratikkshirsagar$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND             CREATED
      STATUS                          PORTS
                                                           NAMES
8572bd1c7074        gitlab/gitlab-ce:latest        "/assets/wrapper"   2 weeks ago
      Up About a minute (healthy)   0.0.0.0:30080->30080/tcp, 0.0.0.0:30022->22/
tcp, 0.0.0.0:20080->80/tcp, 0.0.0.0:32768->443/tcp   gitlab
[Pratiks-MBP:Homework pratikkshirsagar$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND             CREATED
      STATUS                    PORTS
                                                   NAMES
8572bd1c7074        gitlab/gitlab-ce:latest        "/assets/wrapper"   2 weeks ago
      Up 3 hours (healthy)   0.0.0.0:30080->30080/tcp, 0.0.0.0:30022->22/tcp, 0.
0.0.0:20080->80/tcp, 0.0.0.0:32768->443/tcp   gitlab
Pratiks-MBP:Homework pratikkshirsagar$
```

Created Jenkins Jobs

Ran Build Successfully on push event

Build #1 (Mar 8, 2018 8:19:10 PM)

Started by GitLab push by Administrator

No changes.

Started by GitLab push by Administrator

Revision: df768f7c076214f23f4d2f9ad0920d27835135c0

- origin/master

Jacoco - Overall Coverage Summary

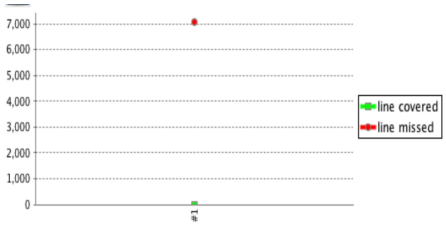
INSTRUCTION	0%	
BRANCH	0%	
COMPLEXITY	0%	
LINE	0%	
METHOD	0%	
CLASS	0%	

Ran Maven build and Jacoco test successfully

localhost:8080/blue/organizations/jenkins/versions-maven-plugin/detail/versions-maven-plugin/1/pipeline

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 15.821 s
[INFO] Finished at: 2018-03-08T20:19:28-06:00
[INFO] Final Memory: 62M/747M
[INFO] ------------------------------------------------------------------------
[JaCoCo plugin] Collecting JaCoCo coverage data...
[JaCoCo plugin] **/**.exec;**/classes;**/src/main/**; locations are configured
[JaCoCo plugin] Number of found exec files for pattern **/**.exec: 0
[JaCoCo plugin] Saving matched execfiles:
[JaCoCo plugin] Saving matched class directories for class-pattern: **/classes:
[JaCoCo plugin]  - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/target/classes 111 files
[JaCoCo plugin] Saving matched source directories for source-pattern: **/src/main/**:
[JaCoCo plugin] WARNING: You are using directory patterns with trailing /, /* or /** . This will most likely multiply the copied files in your build directory. Check the list below and ignore this warning if you know what you are doing.
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/it-set-010-issue-198/fake-api/src/main 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/it-set-010-issue-198/fake-api/src/main/java 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/it-set-010-issue-198/fake-api/src/main/java/org 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/it-set-010-issue-198/fake-api/src/main/java/org/eclipse 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/it-set-010-issue-198/fake-api/src/main/java/org/eclipse/jetty 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/setup-provide-rules-in-jar/src/main 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/setup-provide-rules-in-jar/src/main/resources 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/setup-provide-rules-in-jar/src/main/resources/package 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/setup-provide-rules-in-jar/src/main/resources/package/foo 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/it/setup-provide-rules-in-jar/src/main/resources/package/foo/bar 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions 83 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions/api 11 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions/change 9 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions/ordering 10 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions/rewriting 1 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/java/org/codehaus/mojo/versions/utils 8 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/mdo 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/resources 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/resources/META-INF 0 files
[JaCoCo plugin] - /Users/Shared/Jenkins/Home/workspace/versions-maven-plugin/src/main/resources/META-INF/m2e 0 files
[JaCoCo plugin] Loading inclusions files..
[JaCoCo plugin] inclusions: []
[JaCoCo plugin] exclusions: []
[JaCoCo plugin] Thresholds: JacocoHealthReportThresholds [minClass=0, maxClass=0, minMethod=0, maxMethod=0, minLine=0, maxLine=0, minBranch=0, maxBranch=0, minInstruction=0, maxInstruction=0, minComplexity=0, maxComplexity=0]
[JaCoCo plugin] Publishing the results..
[JaCoCo plugin] Loading packages..
[JaCoCo plugin] Done.
[JaCoCo plugin] Overall coverage: class: 0, method: 0, line: 0, branch: 0, instruction: 0
Finished: SUCCESS
```

Sample Jacoco Test Coverage Report



Overall Coverage Summary

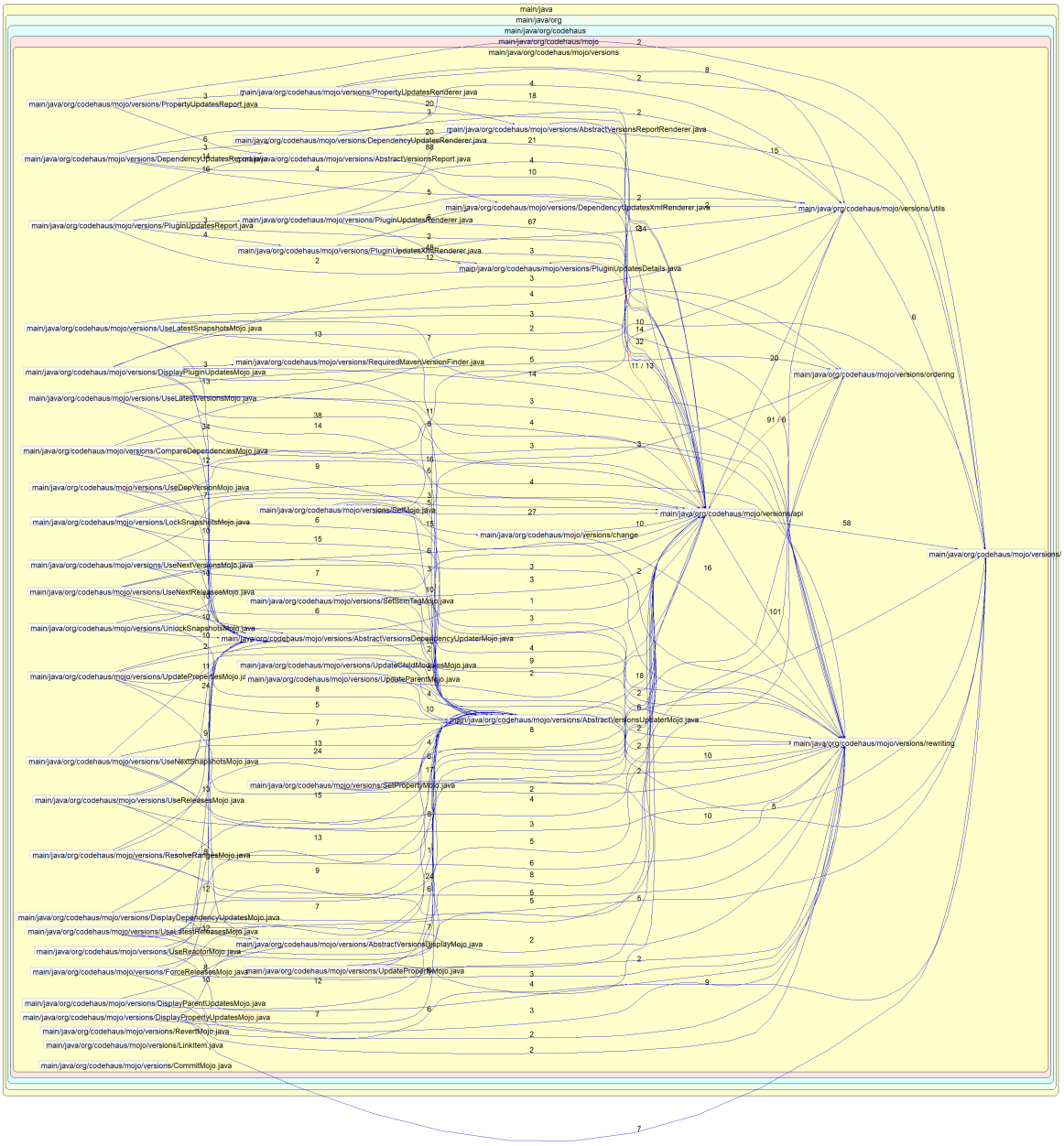
name	instruction	branch	complexity	line	method	class
all classes	0% **M:** 33469 **C:** 0	0% **M:** 3887 **C:** 0	0% **M:** 2736 **C:** 0	0% **M:** 7077 **C:** 0	0% **M:** 786 **C:** 0	0% **M:** 100 **C:** 0

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
org.codehaus.mojo.versions	**M:** 19435 **C:** 0 0%	**M:** 2002 **C:** 0 0%	**M:** 1328 **C:** 0 0%	**M:** 4103 **C:** 0 0%	**M:** 327 **C:** 0 0%	**M:** 47 **C:** 0 0%
org.codehaus.mojo.versions.api	**M:** 7973 **C:** 0 0%	**M:** 1020 **C:** 0 0%	**M:** 739 **C:** 0 0%	**M:** 1623 **C:** 0 0%	**M:** 228 **C:** 0 0%	**M:** 19 **C:** 0 0%
org.codehaus.mojo.versions.change	**M:** 644 **C:** 0 0%	**M:** 72 **C:** 0 0%	**M:** 73 **C:** 0 0%	**M:** 125 **C:** 0 0%	**M:** 37 **C:** 0 0%	**M:** 8 **C:** 0 0%
org.codehaus.mojo.versions.model	**M:** 258 **C:** 0 0%	**M:** 6 **C:** 0 0%	**M:** 35 **C:** 0 0%	**M:** 82 **C:** 0 0%	**M:** 32 **C:** 0 0%	**M:** 3 **C:** 0 0%
org.codehaus.mojo.versions.model.io.xpp3	**M:** 1141 **C:** 0 0%	**M:** 160 **C:** 0 0%	**M:** 120 **C:** 0 0%	**M:** 252 **C:** 0 0%	**M:** 40 **C:** 0 0%	**M:** 3 **C:** 0 0%
org.codehaus.mojo.versions.ordering	**M:** 2327 **C:** 0 0%	**M:** 393 **C:** 0 0%	**M:** 260 **C:** 0 0%	**M:** 537 **C:** 0 0%	**M:** 58 **C:** 0 0%	**M:** 12 **C:** 0 0%
org.codehaus.mojo.versions.rewriting	**M:** 952 **C:** 0 0%	**M:** 128 **C:** 0 0%	**M:** 91 **C:** 0 0%	**M:** 156 **C:** 0 0%	**M:** 27 **C:** 0 0%	**M:** 1 **C:** 0 0%
org.codehaus.mojo.versions.utils	**M:** 739 **C:** 0 0%	**M:** 106 **C:** 0 0%	**M:** 90 **C:** 0 0%	**M:** 199 **C:** 0 0%	**M:** 37 **C:** 0 0%	**M:** 7 **C:** 0 0%

co/#

Dependency Graph



Post running Understand Analyses

```
Generating Understand Reports for javacpp-presets
Cloning into 'docker-maven-plugin'...
remote: Counting objects: 3960, done.
remote: Compressing objects: 100% (1517/1517), done.
remote: Total 3960 (delta 1939), reused 3960 (delta 1939)
Receiving objects: 100% (3960/3960), 692.97 KiB | 14.44 MiB/s, done.
Resolving deltas: 100% (1939/1939), done.
Generating Understand Reports for docker-maven-plugin
Cloning into 'shopping-management-system'...
remote: Counting objects: 1475, done.
remote: Compressing objects: 100% (729/729), done.
remote: Total 1475 (delta 596), reused 1475 (delta 596)
Receiving objects: 100% (1475/1475), 4.25 MiB | 24.85 MiB/s, done.
Resolving deltas: 100% (596/596), done.
Generating Understand Reports for shopping-management-system
Cloning into 'javacv'...
remote: Counting objects: 6390, done.
remote: Compressing objects: 100% (1068/1068), done.
remote: Total 6390 (delta 4374), reused 6390 (delta 4374)
Receiving objects: 100% (6390/6390), 3.69 MiB | 22.52 MiB/s, done.
Resolving deltas: 100% (4374/4374), done.
Generating Understand Reports for javacv
Understand reports have been generated in their respective project roots.
```