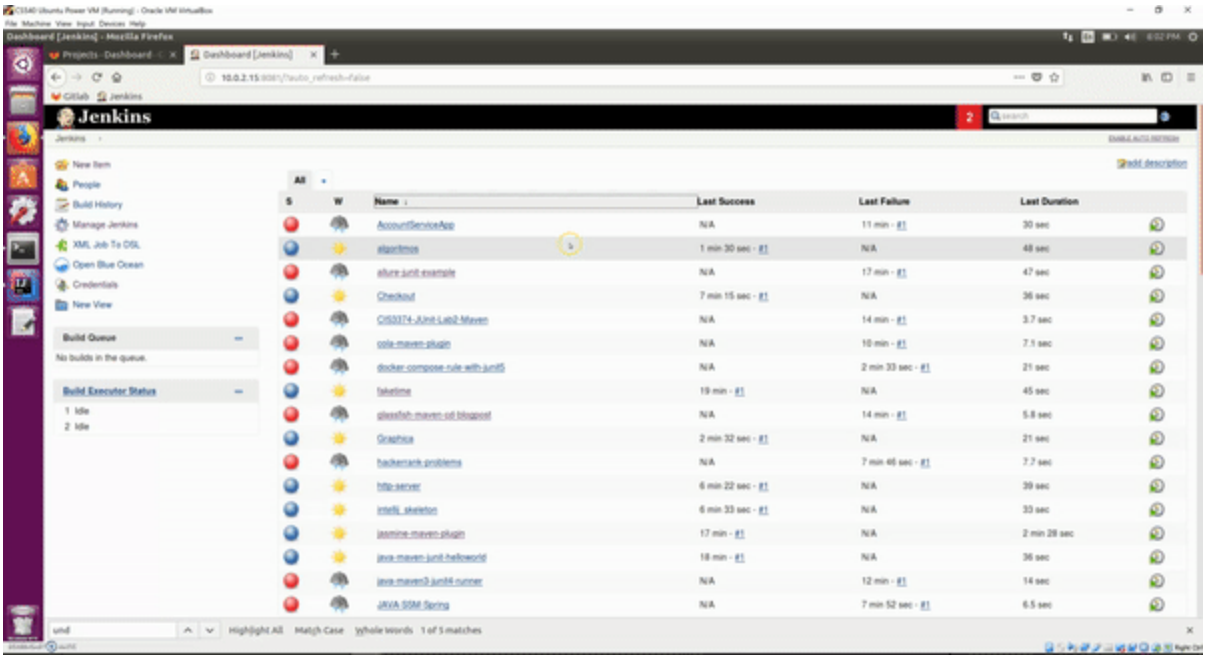
Authors: Dhananjay Gupta,Viren Mody, Guillermo Rojas Hernandez

Project Description:

This project simulates the DevOps environment from developers pushing their code to automated building, testing and analyzing of code. Our Java project simulates this process by downloading repositories from GitHub, uploading each repository to a local GitLab server, which triggers the build process and code analysis in the locally installed Jenkins server. Our Jenkins server configuration built, tested and analyzed project code using JaCoCo and Understand. Our implementation of the DevOps environment focused on Java Maven applications containing unit tests and JaCoCo support.

The project is implemented in Java and built with Gradle. GitHub, GitLab, and Jenkins Java APIs are used to access GitHub.com, the local GitLab server, and the local Jenkins server from the Main.java file. The JGit Java library is used for git repository management within our project. These dependencies are documented in the build.gradle file.

We were able to successfully perform a stress test of our Java application by pushing 54 Github projects through our DevOps simulation without error. From the execution of our application to the last job being built in Jenkins, it took approximately 20 minutes. 25 out of the 54 applications built successfully some of which had included JaCoCo code coverage, unit testing results, and produced Understand code analysis. The remaining 29 failed for a variety of reasons such as missing pom.xml build file, failed unit tests, missing plugins, etc. Please note that our application only pulls the first 100 repository results of our GitHub.com query due to GitHub API limitations. Despite this GitHub API limitation, during our stress test we found that 100 repositories was enough to produce a sufficient number of successful builds.



Configuration Instructions

Please follow these steps:

Machine/Environment

- VirtualBox Virtual Machine
- 8 GB RAM
- Ubuntu 16.04 64-Bit
- Virtual Disk created on setup 50GB (VDI - Virtual Disk Image)
- General Settings > Advanced > Bidirectional for both sharing clipboard and drag and drop
- System Settings > Processor > 4 Cores
- System Settings > Acceleration > Enable VT-x/AMD-V, Nested Paging,
- Display Settings > Video Memory > 128mb and Enable 3D Acceleration

Initial Setup/ Configuration

- Install Oracle JDK 8 (NOT 9)
- Install IntelliJ

- Install Gitlab
    - Local installation, not with docker,
    - Used this tutorial https://about.gitlab.com/installation/#ubuntu
    - Set `EXTERNAL_URL="http://10.0.2.15"` during install. This is the IP for our virtual machine.
    - Default `username: root`
    - `Password: rootroot`
- Install Jenkins
    - Local installation, not with docker,
    - Used this tutorial https://jenkins.io/doc/book/installing/#debian-ubuntu
    - Gitlab is running on `http://10.0.2.15` so Jenkins needs to run on a different port (`i.e. http://10.0.2.15:8081`)
        - Stop Jenkins server: `sudo service jenkins stop`
        - Change `HTTP_PORT=8080` to `HTTP_PORT=8081` in `/etc/default/jenkins`
        - Start Jenkins server: `sudo service jenkins start`
    - please Follow rest of the tutorial to unlock Jenkins, please install suggested plugins
    - Create First Admin User: `Username: admin Password: admin`
- Configure Jenkins
    - Manage Jenkins > Plugins
        - Gitlab
        - Gitlab Hook
        - JaCoCo
        - JUnit
        - Test In Progress
        - Workspace Cleanup
    - Manage Jenkins > Configure Global Security
        - **Uncheck** `Enable security`
        - In section CSRF Protection, **Uncheck** "`Prevent Cross Site Request Forgery Exploits`"

    - Manage Jenkins > Configure System
        - Home Directory should be `/var/lib/jenkins`
        - Jenkins URL should be `http://10.0.2.15:808`

- Gitlab and Jenkins Integration
    - Used this tutorial https://www.swtestacademy.com/jenkins-gitlab-integration/
    - Gitlab > Settings > Access Tokens
        - To add a Jenkins API Access Token, give it a name, and click on all three (api, read_user, sudo) under Scopes. Copy and store the API Token down because it will not be accessible later.
    - Jenkins > Manage Jenkins > Configure System > Gitlab
        - Connection name: GitlabConnection
        - Gitlab host URL: http://10.0.2.15/
        - Credentials > Click on Add > Click on Jenkins
        - Set Kind to Gitlab API Token
        - Enter in the Jenkins API Token you created in Gitlab earlier
        - Enter in ID and Description
        - Now set Credentials to the credentials just created (Gitlab API token)
        - Click on Test Connection to see if it worked
    - From Jenkins main server page > Click on Credentials in the Menu on the Left > Click on System (right below Credentials) > On the right, click on `Global credentials (unrestricted)` > Click on Add Credentials shown on the left. Here add your Gitlab username (i.e. root) and Gitlab password (i.e. rootroot).
    - Install Maven, Gradle, Git

- Install Understand
    - Download the software at Download, unzip the folder, add the path by editing the .bashrc file
        - through home `$ gedit .bashrc`

        - To the bottom of the file, add line `export PATH=$PATH:/location of the unzipped scitools folder/bin/linux64`example: export PATH=$PATH:/home/dhananjay/Downloads/scitools/bin/linux64
        - run `$ source .bashrc` to include the changes OR logout and login again for the changes to take effect.
        - run Understand: `$ understand` and make sure to enter the license key and see if it is up and running.

Building Project

- Install gradle
- Clone project repository from Bitbucket into directory:
    - git
      clone https://drmark@bitbucket.org/guillermokrh/viren_mody_guillermo_rojas_hernandez_dhananjay_gupta_hw1.git
- Edit `Main.java` file with text editor or Intellij
    - Intellij instructions
        - Open up Intellij
        - In the welcome screen, select "Open"
        - Navigate to the the viren_mody_guillermo_rojas_hernandez_dhananjay_gupta_hw1/project1 folder and select "Open"
        - The "Import Project From Gradle" window should pop up.
        - Select the following options:
            - The gradle project path should be the viren_mody_guillermo_rojas_hernandez_dhananjay_gupta_hw1/project1 path
            - Select “Use auto-import”
            - Select “Create separate module per source set”
            - Select “Use default gradle wrapper”
            - For Gradle JVM, use the 1.8 Java version 1.8 JDK
            - Project Format: “.ipr (file based)”
        - Select OK

- Open up Main.java file and modify the variables under the TODO comments in both the customJenkinsJobXML function and the Main function
    - `customJenkinsJobXML()`
    - set `UNDERSTAND_PATH` to the location of the Understand installation, example: “/home/virenmody/Downloads/scitools/bin/linux64/”
    - set `CLONED_REPOS_BASE_PATH` to the directory where you want cloned repositories installed locally, example: “/home/virenmody/ClonedRepos/”
    - `Main()`
    - set `GITLAB_URL` to IP address of the GitLab server, example: "http://10.0.2.15"
    - set `GITLAB_API_ACCESS_TOKEN` to the access token from GitLab, example: “RDtCnzMezmjpih4u6VDm”
    - set `GITLAB_USERNAME` to username used to access Gitlab server, example: “root”
    - set `GITLAB_PASSWORD` to password used to access Gitlab server, example: “rootroot”
    - set `JENKINS_URL` to Jenkins server url from configuration, Example: "http://10.0.2.15:8081"
    - Set `JENKINS_USERNAME` to Jenkins server username, Example: “admin”
    - Set `JENKINS_PASSWORD` to Jenkins server password, Example: “admin”
    - Set `LOCAL_PATH` to the directory you want to save cloned repos
    - Select Build Program
    - Select Run

Analysis Report

The analysis report is being generated at the particular job's folder under the jenkins workspace directory as a `.txt` and html files on your local machine. Kindly, visit the folders to access the reports. Example: under the /var/lib/jenkins/workspace/"JobName". The job name is the name of the Repository being analyzed.