CS480 Software Engineering
Assignment 8 - Continuous Integration

Due Date
Sunday, February 22, 2015

Score
5

Questions and Directions

In this assignment, you will be setting up the continuous integration pipeline for your application using Jenkins, so that the version control, testing, build, release and deployment can be fully automated.

To submit the assignment, please notify me by email that your Jenkins job has been configured successfully. Please send me **1) the URL to your Jenkins job**, such as http://jenkins.cs480yusun.io/jenkins/job/demo-iphoto-web/; **2) the DNS name of your server**, such as http://ec2-54-67-30-104.us-west-1.compute.amazonaws.com:8080/

Please email to both yusun@cpp.edu and yuchinghsu@cpp.edu

0. Build Your Project with Maven in Command-Line Mode

You should be able to run `mvn package` in your project folder and have the whole project built correctly. The final deployable artifact for your project should be the `cs480-1.0.jar` (or other jar file if you use a different name) file in the `target` directory.

*****Note:** once you add unit tests in Assignment 6, you should make sure that your tests can be built and passed with `mvn package`. Particularly, you should be careful about any of the hard-coded test file paths, or the platform-dependent paths (e.g, C:\Users\...). These could cause problems later when you try to use Jenkins to build your project.

1. Prerequisite

- You should have finished Assignment 7 and have a AWS EC2 server up and running.
- You should know the public DNS name of your server, and know how to login to your server with ssh or Putty.
- You need to have an account in Jenkins (username: cs480 password: cs480)
- The Jenkins is located at http://jenkins.cs480.yusun.io/jenkins/

2. Create a new job in Jenkins for your project

Login to Jenkins and choose “New Item”. You will need to create a Maven Project with a unique name (you should name your item with your name or id).

You need to configure our project with the following key options:

1. Choose the “Git” project in “Source Code Management”, and configure the Repository URL for your Git project. The URL can be found in your GitHub project page and is the one you used to git clone.
2. Choose both “Build whenever a SNAPSHOT dependency is built” and “Build when a change is pushed to GitHub” in “Build Triggers”
3. Configure the “Goals and options” with “package” in “Build”
4. Choose “Run only if build succeeds” in “Post Steps”
5. Save

3. Build the project

Click on “Build Now” in Jenkins, you should be able to see Jenkins starts to build your project.

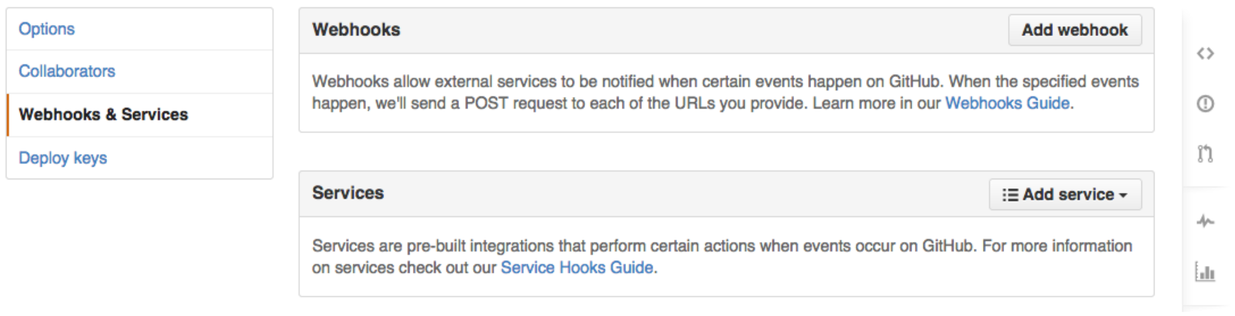
Please check the result of the build. If it fails, go to the build and check “Console Output”. You need to check what the issue is and fix it.

*****Note: your tests and your code may not be built successfully for various reasons. Please read the error messages carefully and fix those by pushing new code to your project and try to build it again. You should also be aware that the Jenkins is running in Linux with JDK7, so the environment may not be the same as your development environment. You need to figure out how to solve the issue when the building process is failed in Jenkins.**

You cannot move to the following steps unless you have your project built correctly.

4. Automated push and build with GitHub and Jenkins

Go to your GitHub project settings page, and choose “Webhooks & Services”:



Webhooks allow external services to be notified when certain events happen on GitHub. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

Services are pre-built integrations that perform certain actions when events occur on GitHub. For more information on services check out our Service Hooks Guide.

Click on "Add webhook", and fill in the following information:

- Payload URL: http://jenkins.cs480.yusun.io/jenkins/github-webhook/
- Content type: application/x-www-form-urlencoded
- "Just the push event"
- Finally, "Add webhook"

To verify all of this, please commit and push a minor change (e.g., change your README file) and see if the build can be automatically triggered in Jenkins.

5. Add post-execution shell script

To automate the deployment process, **you need to configure the key in your own EC2 server so that the Jenkins server can have the access to your EC2 server**. In order to do this, you need to first login to the terminal of your EC2 server (see Assignment 7). Then, edit the file at `.ssh/authorized_keys` with the following command (feel free to use other editors):

```
vim ./.ssh/authorized_keys
```

Then, copy and paste the public key from the following file (add one extra line to paste the whole thing, **DON'T OVERWRITE THE OLD KEY, just add a new line for the new key**), and save the file:

https://s3-us-west-1.amazonaws.com/cs480/key.txt

Now, go back to Jenkins. In your Jenkins job configuration page, you can "Add post-build step", and then choose "Execute shell". Copy and paste the following commands (there are 3 command lines total, when you copy directly from this PDF file, new lines might be created, so make sure you merge the lines together, or you can find all the commands at https://s3-us-west-1.amazonaws.com/cs480/commands.txt), but you need to replace the DNS name (bold) with your own EC2 DNS name, as well as the jar file name if yours is different:

```
scp -o "StrictHostKeyChecking no" ./target/cs480-1.0.jar
ec2-user@ec2-54-67-13-222.us-west-1.compute.amazonaws.com:~/
```

```
ssh -o "StrictHostKeyChecking no"
ec2-user@ec2-54-67-13-222.us-west-1.compute.amazonaws.com "killall java" ||
true

ssh -o "StrictHostKeyChecking no"
ec2-user@ec2-54-67-13-222.us-west-1.compute.amazonaws.com 'nohup java -jar
/home/ec2-user/cs480-1.0.jar >/dev/null 2>&1' &
```

Save your change and trigger a new build, followed by verifying if the build can be succeeded.

Finally, push a new code change to GitHub and see if a successful build + deployment can be done automatically. If so, great job!