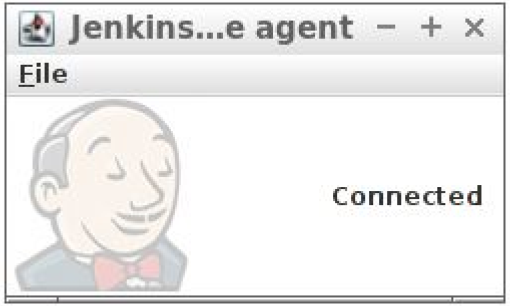
Practical Workshop

Initialisation

- Go to localhost:8080 to access Jenkins.
- Click ‘Manage Jenkins’ -> ‘Manage Nodes’ -> ‘Build Executor’
- Click on ‘Launch’ to launch the build agent (OK all of the pop up windows)
- Window on bottom right will pop up.
- Leave this open from now on.
- You now have a build agent running as well as the server itself.



Connected

Task 1 - Setting up github.

- There is a git repository at https://github.com/SDiserens/Continuous-Integration-Lab.
- Access this page whilst logged into your github account.
- Click on the fork button and make a fork of it on your own account.

Task 2 - “Green Balls are Better than Blue Balls.”

- By default Jenkins uses blue as an indicator to mean success, this is confusing and less satisfying than a green indicator!
- Navigate to ‘Manage Jenkins’ -> ‘Manage Plugins’ -> ‘Available’
- Install the ‘Green Balls’ plugin.
- Click the box to restart Jenkins once plugin is installed.
- Installing any plugin is as simple as this!
- We will see Green Balls in action later on!

Task 3 - Checking Out Code from Github.

- Create a new ‘Freestyle Project’ on the Jenkins dashboard (use ‘New Item’ link)
- On the project configuration page under ‘Source Code Management’ configure Jenkins to check out the repository from your github account.
- Click on “Build Now” on the project page.
- Make sure that the project success ball goes green!
- On the project page click on the latest build number and then ‘Console Output’, ensure that the code is being checked out from github.
- Warning messages are expected on the configuration page - ignore them!

Task 4 - Running Python Code

- On the project configuration page add a build event to execute a shell command, this command should execute FiniteElements.py - jenkins is configured such that your are in the correct directory.
- Click on “Build Now” on the project page.
- Make sure that the project success ball goes green!
- On the project page click on the latest build number and then “console output”, ensure that the python script outputs the expected message.
- If you have a warning of missing libraries then grab Pete or Sam!

Task 5 - Build Code On Commit

- On the project configuration page under “Build Triggers” configure jenkins to build code every time a change is made to the github repository. Configure the schedule to run every minute.
- Make a note of the current latest build number on the project configuration page.
- Make a change to your forked FiniteElements.py file such that it prints out something different, this can be done directly on Github! (No need to check out the code locally)
- Observe that a new build was triggered on the project configuration page.
- Make sure that the project success ball goes green!
- On the project page click on the latest build number and then “console output”, ensure that the python script has output your changes.

Task 6 - Automatically Build at Time Intervals

- Update the “Build Triggers” scheduler to build your code every 15 minutes.
- Observe that a new build was triggered on the project configuration page.
- Make sure that the project success ball goes green!

Task 7 - Run unit testing

- On the project configuration page add a build event to execute a shell command, this command should execute FiniteElementsTest.py.
- The unit testing framework output a results file in a format called J-Unit, we want to use this to display our test results.
- Also update jenkins “Post Build Event” to process the J-Unit test output from the finite elements test code. The path is **/test-reports/*.xml
- Click on “Build Now” on the project page.
- Make sure that the project success ball goes red!
- On the build number page view the test results to find out which test has broken!

Task 8 - Fix the Failing Test!

- Make changes to the FiniteElementsTest.py file in your Github fork such that the failing unit test now runs successfully.
- Ensure that the test results are updated accordingly on the test results page.

Summary

- Bad practices
  - Polling of GitHub every minute.
  - Not splitting jobs across multiple builds.
- Managing more complex jobs with a build script
- In reality we might implement:
  - Code run on commit hook
    - Tests the integration of the change
    - Know immediately if something is broken
  - Run tests at specific times
    - .Tests the functionality of the software
    - Can be scheduled to run overnight when user load is less