Configuring Selenium Grid Across Multiple PCs

Sean McGrath, [sean.mcgrath@tcs.com](mailto:sean.mcgrath@tcs.com)

**Selenium Grid:**

Selenium Grid is a part of the Selenium Suite which allows for running multiple tests across multiple machines, browsers, and operating systems. The tests are run on a single machine called a *hub*, and executed on one or more machines called *nodes*. The benefit of Selenium Grid is that tests can be run on different browsers and operating systems in parallel, and it allows testers to save time in executing a test suite.

**Configuring Selenium Grid with One Hub and One Node**

To start simply, we will configure our grid with one hub and one node. We will refer to these machines as Machine H and Machine N\_0, respectively. Remember there can only be one hub in a grid, as it is the centralized point for loading all tests. Each node is a Selenium instance that executes a test loaded onto the hub. To begin, first we must download the Selenium Standalone Server .jar file on each machine. As of this writing, the current version is 3.0.1.

<http://docs.seleniumhq.org/download/>

To finish install, simply save the .jar file onto each machine’s file system, we will just put ours directly on the C: drive.

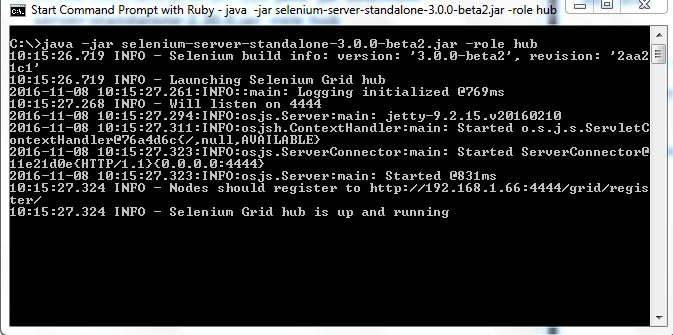
**Launch a hub**

On Machine H, navigate to the root of H’s C: drive via the command prompt.

Enter following command on the command prompt:

java –jar selenium-server-standalone-3.0.1.jar –role hub

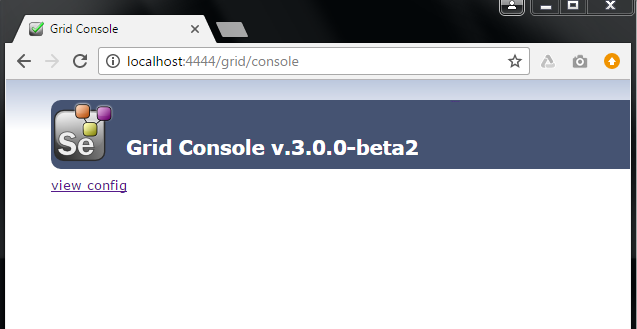
The hub should be launched and your command prompt should look similar to the following:



The hub defaults to run on port 4444, so on Machine H open a browser and navigate to

localhost:4444/grid/console

You should see something like this:



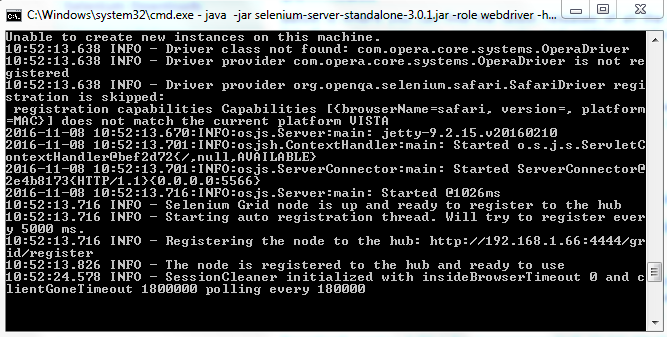
The Hub has been successfully launched!

**Launch a Node**

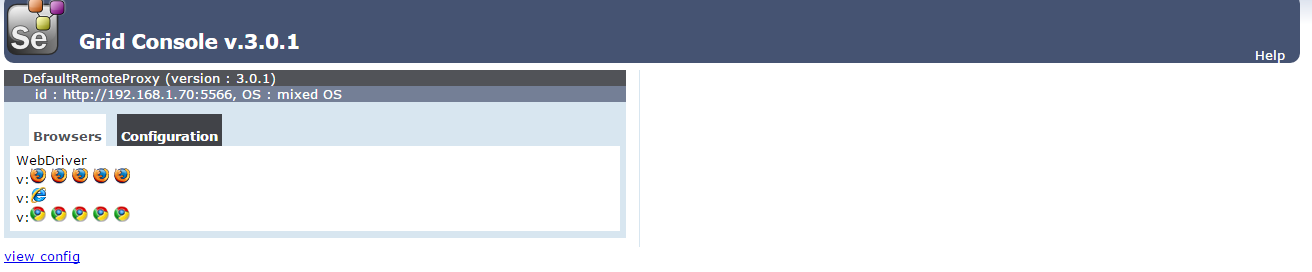
On Machine N\_0, open command prompt and navigate to the root of N\_0’s C: drive. Enter the following command:

java –jar selenium-server-standalone-3.0.1.jar –role webdriver –hub http://<ip-address-of-Machine-H>:4444/grid/register –port 5566

Here we used port 5566, but you may use any free port you desire. If successful, N\_0’s command prompt should look like the following:



Subsequently, this registers the hub, as noted in the above figure. Now, navigate to localhost:4444/grid/console, which should look like this:



**Writing a test**

For this tutorial, we will develop our test in eclipse using Maven and TestNG. First, create a new Maven project:

* File->New->Project->Maven Project
* Select the ‘Create a simple project (skip architype)’ checkbox
* Enter an appropriate Group Id and Artifact ID and select finish (we used ‘selenium-grid-test’ for each)

In the new project, navigate to the pom.xml file, and add the following dependencies:

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.9.10</version>

<scope>test</scope>

</dependency>

<dependency>

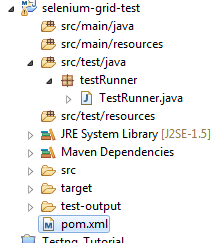
<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

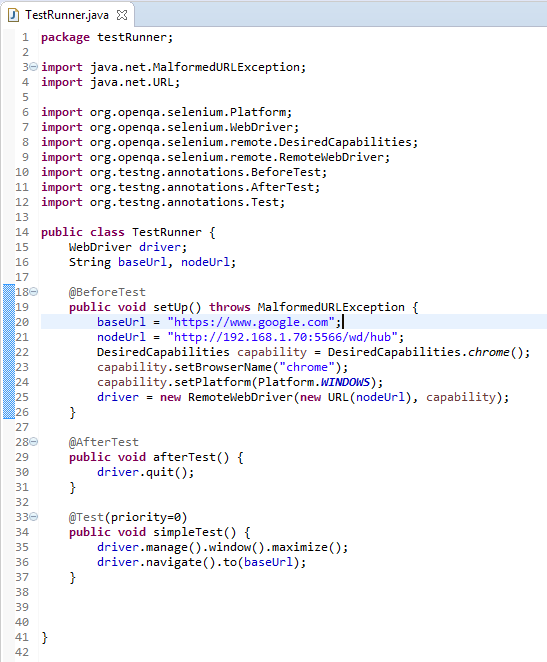
<version>3.0.1</version>

</dependency>

Now create a new package—we named ours ‘testRunner’—inside of src/test/java. In this package create a new java class from where you will execute your test—ours is TestRunner.java. Now your project should have the following architecture:



Add the following code to TestRunner.java, which we discuss in detail below:



Since we are using TestNG, we use the annotations @*BeforeTest*, @*AfterTest*, and @*Test* to control the execution of the test suite, avoiding the need for a *public static void main(String[] args)* method.

The String baseUrl is the URL we will navigate to, and the nodeUrl is the URL of Machine N\_0, which has the form http://<Machine-N\_0-ip-address>:<Machine-N\_0-port>/wd/hub.

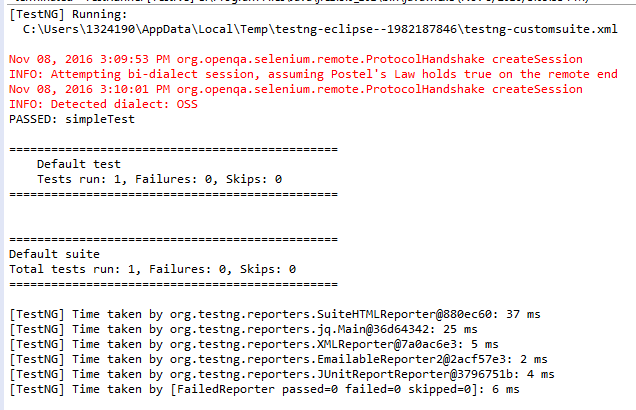
The DesiredCapabilities Object allows us to set the browser and platform to execute our test.

Finally, we set our driver to a new RemoteWebDriver Object, passing it the URL of the node and the capabilities.

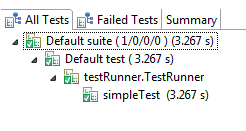
**Execute the test:**

Before executing your test, make sure that Machine N\_0 has chromedriver.exe in the same directory as selenium-server-standalone-3.0.1.jar and that chromedriver.exe is compatible with the version of chrome installed on Machine N\_0.

On Machine H, run TestRunner.java from eclipse as a TestNG Test. On Machine N\_0, you should see a chrome browser be opened and maximized, navigate to [www.google.com](http://www.google.com), and finally close the window. The console output on Machine H should appear similar to this:



And the TestNG output should appear similar to this:



Integrating Test Reporting with ATU Reporter

In order to generate better reports with TestNG, we need to integrate ATU Reporter. The first step is downloading the jar files from

<https://drive.google.com/drive/folders/0B7rZvkq9tkwPMkJlUjJGTkJjOTA>

Download the 5.1.1 jar files, unzip them, and save them to some location on your computer (We simply put them on the C:/ drive).

**Creating a Maven Dependency with Third Party jars**

Because ATU Reporter is not integrated with Maven—and since we are using a Maven project—we need to define our own Maven dependencies for the two ATU jars.

Open a command prompt and navigate to the location of the ATU Reporter jar files.

Enter the following two commands:

mvn install:install-file –Dfile=ATUReporter\_Selenium\_testNG\_5.1.1.jar –DgroupId=atu \

-DartifactId=reporter –Dversion=5.1.1 –Dpackaging=jar

mvn install:install-file –Dfile=ATUTestRecorder\_2.1.jar –DgroupId=atu \

-DartifactId=recorder –Dversion=5.1.1 –Dpackaging=jar

Now add the dependencies to the pom.xml file and save to build the project.

<dependency>

<groupId>atu</groupId>

<artifactId>reporter</artifactId>

<version>5.1.1</version>

</dependency>

<dependency>

<groupId>atu</groupId>

<artifactId>recorder</artifactId>

<version>2.1</version>

</dependency>

**Configuring test classes for ATU:**

All we need to do in our two test classes (GoogleTestRunner.java and GoogleSearchTestRunner.java) is to add Listeners and set a system property:

Add the following lines of code to the top of each class declaration:

@Listeners({ ATUReportsListener.**class**, ConfigurationListener.**class**, MethodListener.**class** })

**public** **class** GoogleTestRunner {

**public** **static** RemoteWebDriver *driver*;

**public** **static** String *appURL* = "https://www.google.com";

{//File location for atu

System.*setProperty*("atu.reporter.config", "C:\\Users\\1324190\\eclipseWorkspace\\Copy of SeleniumGridParrallel\\atu.properties");

}

We also need to add the following import statements to each file:

**import** atu.testng.reports.ATUReports;

**import** atu.testng.reports.listeners.ATUReportsListener;

**import** atu.testng.reports.listeners.ConfigurationListener;

**import** atu.testng.reports.listeners.MethodListener;

**import** atu.testng.reports.logging.LogAs;

**import** atu.testng.selenium.reports.CaptureScreen;

**import** atu.testng.selenium.reports.CaptureScreen.ScreenshotOf;

**Editing atu.properties file:**

In the package explorer, open atu.properties. We need to edit lines 2 and 8 to appear like the following:

2: atu.reports.dir=C:/Users/1324190/eclipseWorkspace/SeleniumGridParrallel/test-output/atu

8: atu.proj.header.logo=C:/Users/1324190/eclipseWorkspace/Copy of SeleniumGridParrallel/test-output/bullet\_point.png

**Editing testNG.xml**

Add the following listeners to testng.xml in order to generate a single ATU report for each test class executed.

<listeners>

<listener class-name=*"atu.testng.reports.listeners.ATUReportsListener"*></listener>

<listener class-name=*"atu.testng.reports.listeners.ConfigurationListener"*></listener>

<listener class-name=*"atu.testng.reports.listeners.MethodListener"*></listener>

</listeners>

</suite> <!-- Suite -->

You are now ready to generate ATU reports! Execute your test suite and then open

test-output->atu->index.html to view the report.