Configuring Parallel Tests across Machines Using Selenium Grid Tutorial

Sean McGrath ([sean.mcgrath@tcs.com](mailto:sean.mcgrath@tcs.com)), Aaron Wingo ([aaron.wingo@tcs.com](mailto:aaron.wingo@tcs.com))

**Introduction to Selenium Grid**

Selenium Grid allows testers to run tests on multiple machines from a single central hub. This is useful for testing across platforms and browsers, or for speeding up test execution by running tests in parallel. This tutorial will take you through configuration of a simple test suite which runs parallel tests on two different PCs, and in different browsers.

**Requirements**

* Eclipse (We are using Neon 4.6.0)
* Eclipse Maven plugin
* Eclipse TestNG plugin
* Selenium Standalone Server 3.0.1 jar file (<http://www.seleniumhq.org/download/>)
* chromedriver.exe
* IEDriverServer.exe

**Configuring a Grid**

We will run our tests on two PCs, Machine H and Machine A. Machine H will be our hub and also run a node. Machine A will be running a single node. To install Selenium Standalone Server, simply download the .jar file and save it to some location on your computer (for this tutorial, selenium-server-standalone.jar is saved to the C:/ drive). This must be done on both Machine H and Machine A.

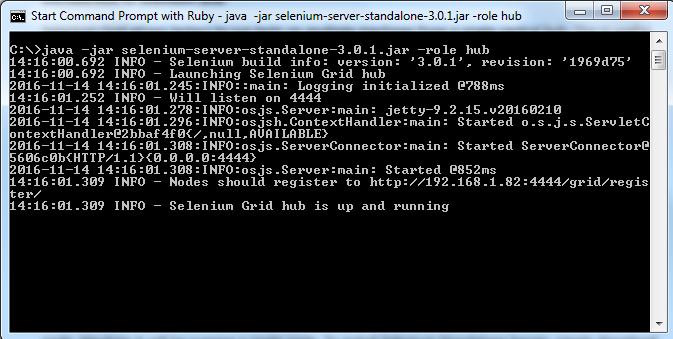
Open a Windows command prompt and navigate to the directory containing selenium-server-standalone.jar. We will now set up our hub:

On Machine H, enter the following command into the command prompt:

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

This starts a grid on Machine H, using the default port setting of 4444. You can always change the default port using the –port option.

The command prompt should look similar to the following:



Now, we will configure a node on Machine H:

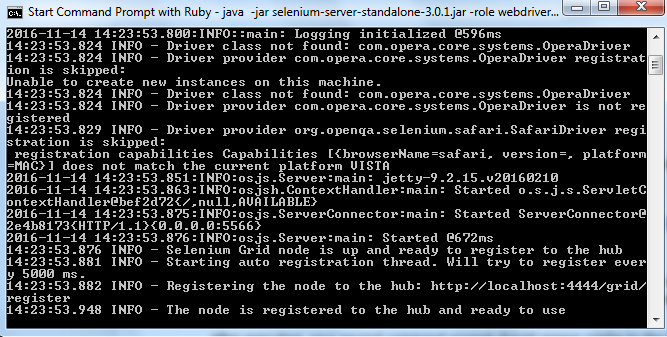
Open a separate command prompt on Machine H, navigate to the location of the

selenium-server-standalone.jar. Enter the following command:

java –jar selenium-server-standalone-3.0.1.jar –role webdriver –hub [http://localhost:4444/grid/register -port 5566](http://localhost:4444/grid/register%20-port%205566)

Feel free to use any open port, we are using 5566. Note we must specify the port of the hub on localhost; so if you did not use the default settings make sure your port numbers match.

After executing, your second command prompt should appear similar to the following:

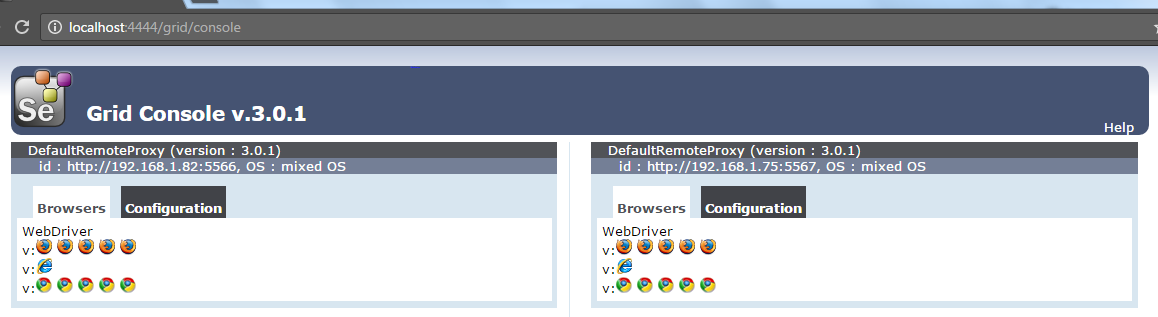


Now, on the remote machine (Machine A) navigate to C:/ drive via command prompt and enter following command:

java –jar selenium-server-standalone-3.0.1.jar –role webdriver –hub [http://192.168.1.82:4444/grid/register -port 5567](http://192.168.1.82:4444/grid/register%20-port%205567)

Note that you must know the domain name of Machine H, which in our case it is 192.168.1.82. You can also specify any open port you wish, here we use 5567 for clarity in comparison with the node on Machine H.

After configuring the node on Machine A, open up a browser on Machine H and navigate to <http://localhost:4444/grid/console>. You should see something similar to the following:



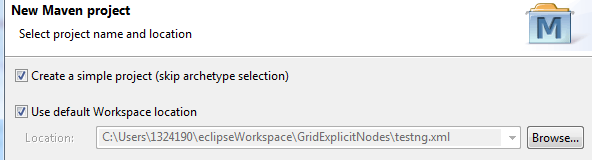
Note the two nodes and their ip addresses, we have successfully set up our grid! You will notice the browser symbols in each node, this tells us the maximum number of webdrivers that each node can run at one time. Hovering your mouse over one of these will also tell you the platform that they will be running on.

**Creating a Maven Project in Eclipse**

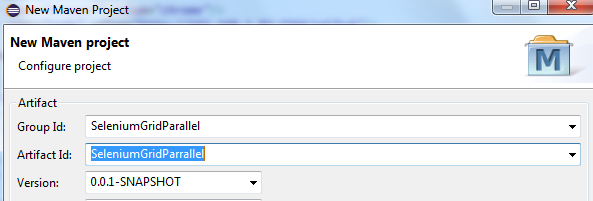
Launch Eclipse and select File->New->Project

Select ‘Maven Project’ and click ‘Next’.

Select the ‘Create a simple project’ checkbox as shown below and select ‘Next’



Enter a Group ID and Artifact ID, we will use SeleniumGridParallel.



Click ‘Finish’.

**Maven Dependencies**

We must add two Maven Dependencies to our pom.xml file, testng and selenium-java. We have supplied the code for these dependencies to copy into your pom.xml below:

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.9.10</version>

<scope>test</scope>

</dependency>

<dependency>

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

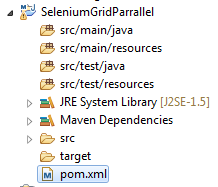
<artifactId>selenium-java</artifactId>

<version>3.0.1</version>

</dependency>

</dependencies>

Click on the ‘save’ button to build the project. You will see the necessary jar files added to ‘Maven Dependencies’ in the project architecture.



Now we will convert our project into a TestNG project. Right click on the project in the package explorer and select TestNG->Convert to TestNG. Click ‘Finish’ and you will see testng.xml added to your project.

**Browser.java**

Create a new package inside of /src/test/java. We will name ours testSuite. Right click on this package in the Project Explorer and create a new Java Class named ‘Browser’. This class will instantiate the RemoteWebDriver object that executes a given test class on a given node. We do this to be able to specify the node that our test should be run on, and specify which browser we will run the test in. The browser and node URL will be passed as parameters in our testng.xml file, but let’s first consider the Browser.java code:

**package** testSuite;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** org.openqa.selenium.remote.DesiredCapabilities;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**public** **class** Browser {

**public** **static** RemoteWebDriver getDriver(String browser, String nodeURL) **throws** MalformedURLException {

**return** **new** RemoteWebDriver(**new** URL(nodeURL), *getBrowserCapabilities*(browser));

}

**private** **static** DesiredCapabilities getBrowserCapabilities(String browserType) {

**switch** (browserType.toLowerCase()) {

**case** "firefox":

System.***out***.println("Opening firefox driver");

**return** DesiredCapabilities.*firefox*();

**case** "chrome":

System.***out***.println("Opening chrome driver");

**return** DesiredCapabilities.*chrome*();

**case** "ie":

System.***out***.println("Opening IE driver");

**return** DesiredCapabilities.*internetExplorer*();

**default**:

System.***out***.println("browser : " + browserType + " is invalid, Launching Firefox as browser of choice..");

**return** DesiredCapabilities.*firefox*();

}

}

}

Now create another java class named GoogleTestRunner.java inside of the testSuite package. The code is given below:

**package** testSuite;

**import** java.net.MalformedURLException;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.AfterClass;

**import** org.testng.annotations.BeforeClass;

**import** org.testng.annotations.Parameters;

**import** org.testng.annotations.Test;

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

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

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

@BeforeClass

@Parameters({ "browser", "node" })

**public** **void** setUp(String browser, String node) **throws** MalformedURLException {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

*driver* = Browser.*getDriver*(browser, node);

*driver*.manage().window().maximize();

}

@AfterClass

**public** **void** tearDown() {

**if**(*driver*!=**null**) {

System.***out***.println("Closing browser");

*driver*.quit();

}

}

@Test

**public** **void** testNavigationToPenFed() {

*driver*.navigate().to(*appURL*);

String pageTitle = *driver*.getTitle();

System.***out***.println("The page title is: " + pageTitle);

}

}

Now create a second test class, GoogleSearchTestRunner.java within suiteRunner.

**package** testSuite;

**import** java.net.MalformedURLException;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.AfterClass;

**import** org.testng.annotations.BeforeClass;

**import** org.testng.annotations.Parameters;

**import** org.testng.annotations.Test;

**public** **class** GoogleSearchTestRunner {

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

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

@BeforeClass

@Parameters({ "browser", "node" })

**public** **void** setUp(String browser, String node) **throws** MalformedURLException {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

*driver* = Browser.*getDriver*(browser, node);

*driver*.manage().window().maximize();

}

@AfterClass

**public** **void** tearDown() {

**if**(*driver*!=**null**) {

System.***out***.println("Closing browser");

*driver*.quit();

}

}

@Test(priority=0)

**public** **void** testNavigationToGoogle() {

*driver*.navigate().to(*appURL*);

String pageTitle = *driver*.getTitle();

System.***out***.println("The page title is: " + pageTitle);

}

@Test(priority=1)

**public** **void** viewProducts() {

WebElement searchButton = *driver*.findElement(By.*name*("q"));

searchButton.sendKeys("selenium server");

searchButton.submit();

}

}

Now we will edit our testng.xml file in order to run our project as a testNG Suite. The code is below. Pay attention to how we are executing tests in parallel, and passing in the browser type and node ip address as parameters for each test.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"tests"*>

<test name=*"Test"*>

<parameter name=*"browser"* value=*"chrome"*/>

<parameter name=*"node"* value=*"http://192.168.1.82:5566/wd/hub"*/>

<classes>

<class name=*"testSuite.GoogleTestRunner"*/>

</classes>

</test> <!-- Test -->

<test name=*"Test2"*>

<parameter name=*"browser"* value=*"chrome"*/>

<parameter name=*"node"* value=*"http://192.168.1.82:5566/wd/hub"*/>

<classes>

<class name=*"testSuite.GoogleSearchTestRunner"*/>

</classes>

</test>

<test name=*"Test3"*>

<parameter name=*"browser"* value=*"ie"*/>

<parameter name=*"node"* value=*"http://192.168.1.75:5567/wd/hub"*/>

<classes>

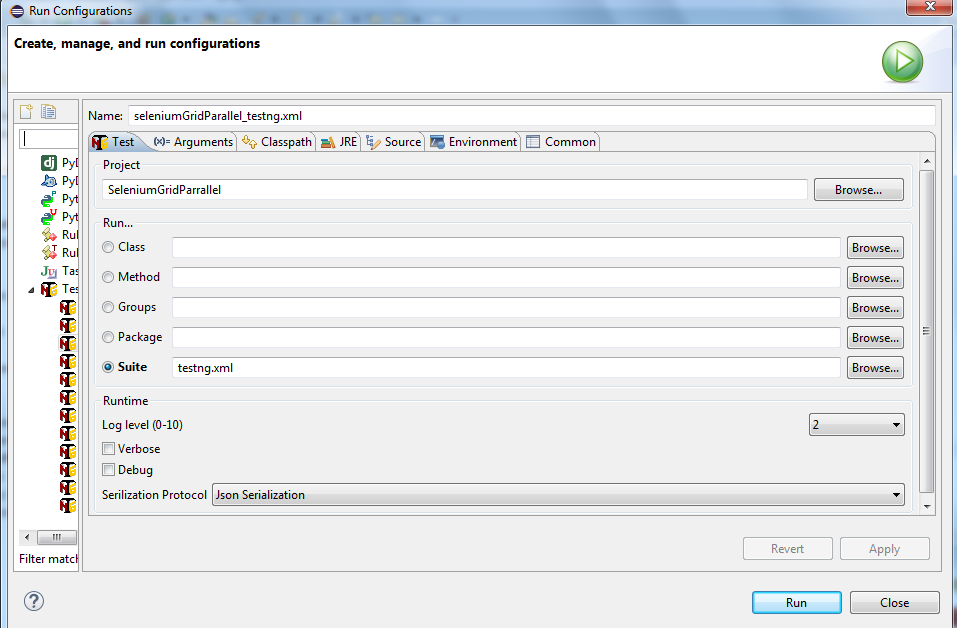
<class name=*"testSuite.GoogleTestRunner"*/>

</classes>

</test>

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

To execute the test right click inside of testng.xml and select Run->Run Configurations. Specify the proper Project and Suite, it should look as follows. Click Run.



If you have been careful in keeping track of your ip addresses and port numbers, and the chromedriver.exe and IEDriverServer.exe are both in the same directory as selenium-server-standalone.jar, then each test should execute in parallel and pass.