

Configuring Selenium Grid Across Multiple PCs

Sean McGrath, 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.

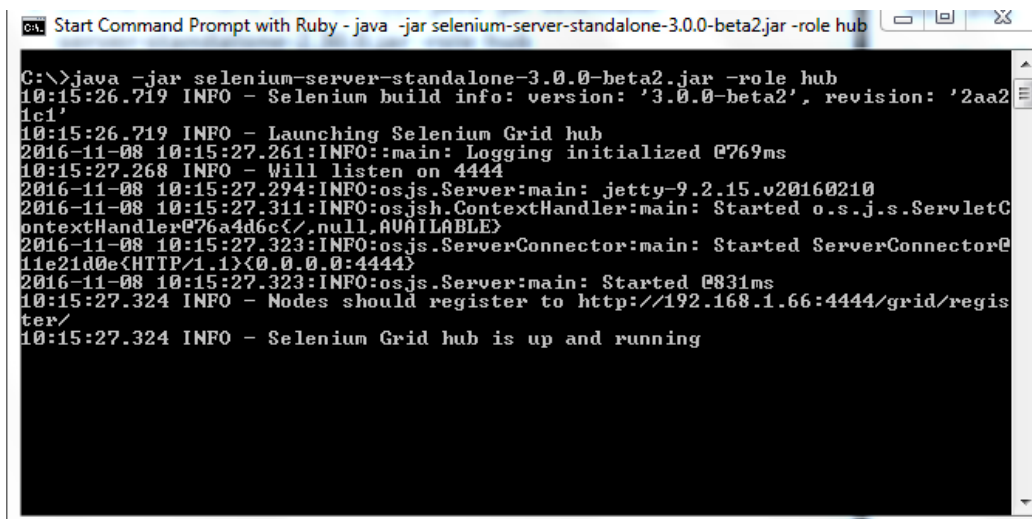
Launch a hub

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:



```
Start Command Prompt with Ruby - java -jar selenium-server-standalone-3.0.0-beta2.jar -role hub
C:\>java -jar selenium-server-standalone-3.0.0-beta2.jar -role hub
10:15:26.719 INFO - Selenium build info: version: '3.0.0-beta2', revision: '2aa21c1'
10:15:26.719 INFO - Launching Selenium Grid hub
2016-11-08 10:15:27.261:INFO::main: Logging initialized @769ms
10:15:27.268 INFO - Will listen on 4444
2016-11-08 10:15:27.294:INFO:osjs.Server:main: jetty-9.2.15.v20160210
2016-11-08 10:15:27.311:INFO:osjs.ContextHandler:main: Started o.s.j.s.ServletContextHandler@76a4d6c{/,null,AVAILABLE}
2016-11-08 10:15:27.323:INFO:osjs.ServerConnector:main: Started ServerConnector@11e21d0e[HTTP/1.1]{0.0.0.0:4444}
2016-11-08 10:15:27.323:INFO:osjs.Server:main: Started @831ms
10:15:27.324 INFO - Nodes should register to http://192.168.1.66:4444/grid/register/
10:15:27.324 INFO - Selenium Grid hub is up and running
```

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

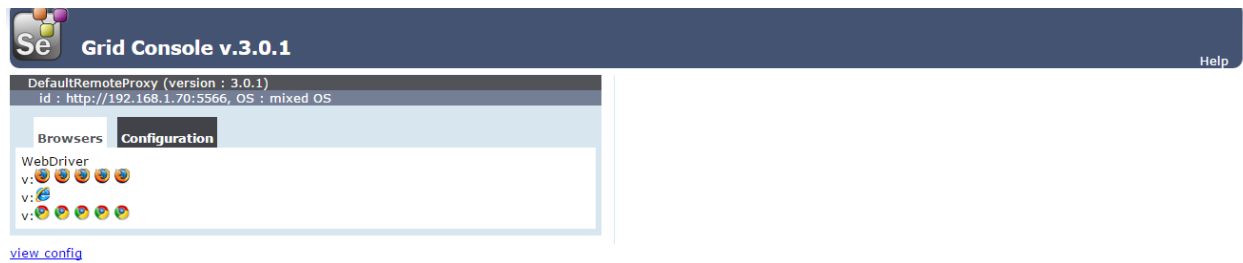
On Machine N_0, open command prompt and navigate to the root of N_0's C: drive. Enter the following command:

```
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:

```
C:\Windows\system32\cmd.exe - java -jar selenium-server-standalone-3.0.1.jar -role webdriver -h...
Unable to create new instances on this machine.
10:52:13.638 INFO - Driver class not found: com.opera.core.systems.OperaDriver
10:52:13.638 INFO - Driver provider com.opera.core.systems.OperaDriver is not registered
10:52:13.638 INFO - Driver provider org.openqa.selenium.safari.SafariDriver registration is skipped:
registration capabilities Capabilities [{browserName=safari, version=, platform=MAC}] does not match the current platform USTA
2016-11-08 10:52:13.670:INFO:osjs.Server:main: Jetty-9.2.15.v20160210
2016-11-08 10:52:13.701:INFO:osjs.ContextHandler:main: Started o.s.j.s.ServletContextHandler@bef2d72[/,null,AVAILABLE]
2016-11-08 10:52:13.701:INFO:osjs.ServerConnector:main: Started ServerConnector@2e4b8173[HTTP/1.1]{0.0.0.0:5566}
2016-11-08 10:52:13.716:INFO:osjs.Server:main: Started @1026ms
10:52:13.716 INFO - Selenium Grid node is up and ready to register to the hub
10:52:13.716 INFO - Starting auto registration thread. Will try to register every 5000 ms.
10:52:13.716 INFO - Registering the node to the hub: http://192.168.1.66:4444/grid/register
10:52:13.826 INFO - The node is registered to the hub and ready to use
10:52:24.578 INFO - SessionCleaner initialized with insideBrowserTimeout 0 and clientGoneTimeout 1800000 polling every 1800000
```

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

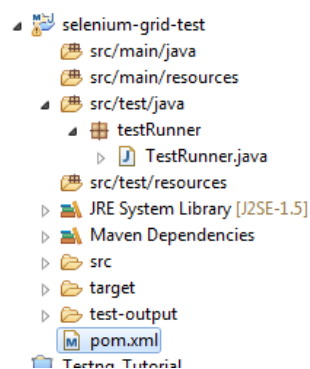
For this tutorial, we will develop our test in eclipse using Maven and TestNG. First, create a new Maven project:

- File->New->Project->Maven Project
- Select the 'Create a simple project (skip archetype)' 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:

```
TestRunner.java
1 package testRunner;
2
3 import java.net.MalformedURLException;
4 import java.net.URL;
5
6 import org.openqa.selenium.Platform;
7 import org.openqa.selenium.WebDriver;
8 import org.openqa.selenium.remote.DesiredCapabilities;
9 import org.openqa.selenium.remote.RemoteWebDriver;
10 import org.testng.annotations.BeforeTest;
11 import org.testng.annotations.AfterTest;
12 import org.testng.annotations.Test;
13
14 public class TestRunner {
15     WebDriver driver;
16     String baseUrl, nodeUrl;
17
18     @BeforeTest
19     public void setUp() throws MalformedURLException {
20         baseUrl = "https://www.google.com";
21         nodeUrl = "http://192.168.1.70:5566/wd/hub";
22         DesiredCapabilities capability = DesiredCapabilities.chrome();
23         capability.setBrowserName("chrome");
24         capability.setPlatform(Platform.WINDOWS);
25         driver = new RemoteWebDriver(new URL(nodeUrl), capability);
26     }
27
28     @AfterTest
29     public void afterTest() {
30         driver.quit();
31     }
32
33     @Test(priority=0)
34     public void simpleTest() {
35         driver.manage().window().maximize();
36         driver.navigate().to(baseUrl);
37     }
38
39
40
41 }
42
```

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, and finally close the window. The console output on Machine H should appear similar to this:

```
[TestNG] Running:
C:\Users\1324190\AppData\Local\Temp\testng-eclipse--1982187846\testng-customsuite.xml

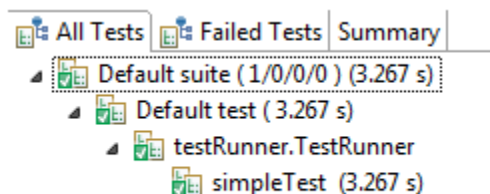
Nov 08, 2016 3:09:53 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Attempting bi-dialect session, assuming Postel's Law holds true on the remote end
Nov 08, 2016 3:10:01 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
PASSED: simpleTest

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====

=====
Default suite
Total tests run: 1, Failures: 0, Skips: 0
=====

[TestNG] Time taken by org.testng.reporters.SuiteHTMLReporter@880ec60: 37 ms
[TestNG] Time taken by org.testng.reporters.jq.Main@36d64342: 25 ms
[TestNG] Time taken by org.testng.reporters.XMLReporter@7a0ac6e3: 5 ms
[TestNG] Time taken by org.testng.reporters.EmailableReporter2@2acf57e3: 2 ms
[TestNG] Time taken by org.testng.reporters.JUnitReportReporter@3796751b: 4 ms
[TestNG] Time taken by [FailedReporter passed=0 failed=0 skipped=0]: 6 ms
```

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/0B7rZvkq9tkwPMkJlUjJGTkIjOTA>

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.