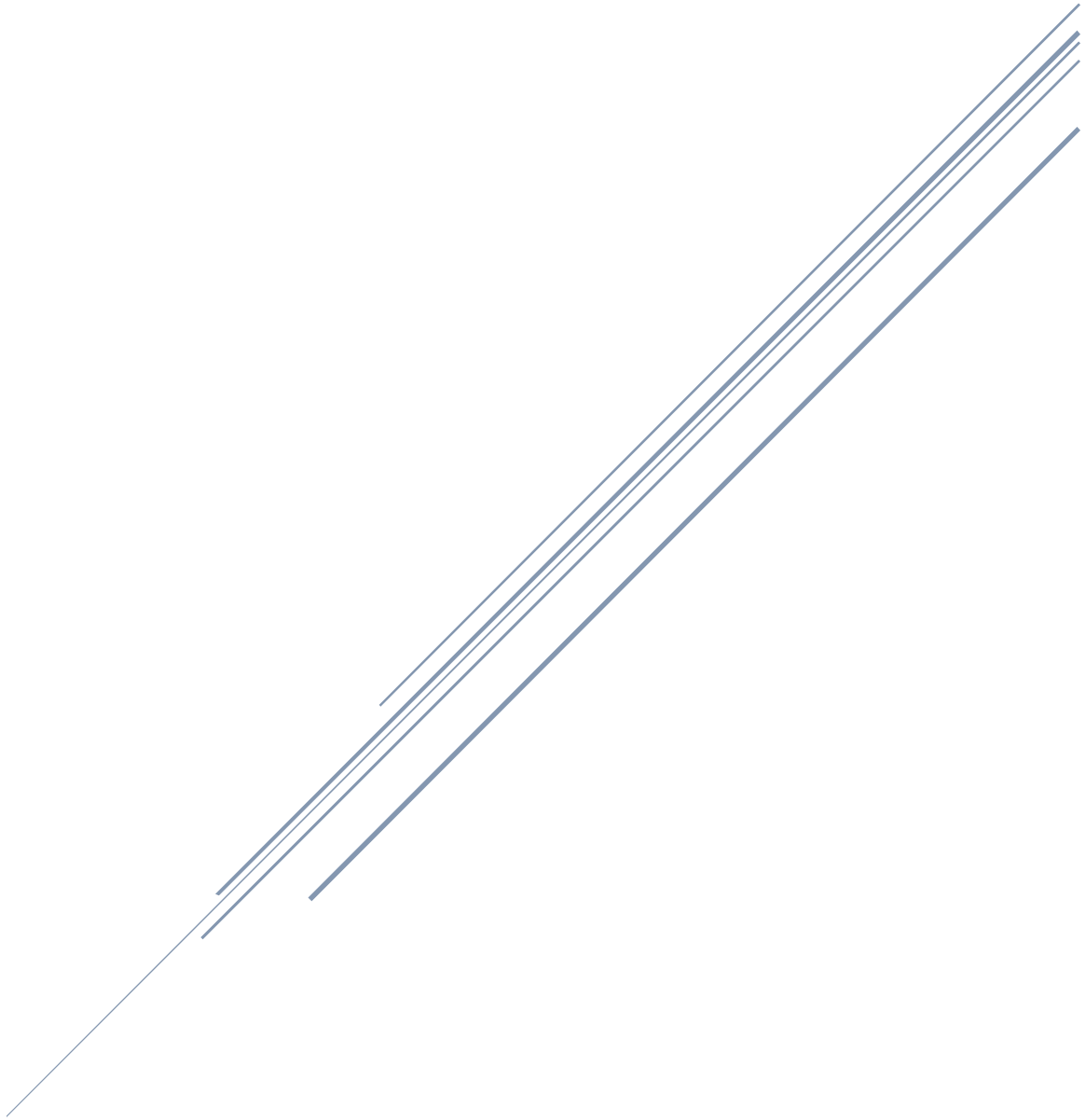


# JMETER – SELENIUM INTEGRATION



2021  
JMETER

## Contents

|  |    |
|--|----|
| JMeter – Selenium WebDriver Integration .....                    | 2  |
| JUnit Request Sampler .....                                      | 2  |
| Step-by-step.....  | 3  |
| General Guidelines .....   | 3  |
| Step-by-Step-GUI .....   | 4  |
| JDBC Sampler.....  | 7  |
| WebDriver Sampler .....  | 8  |
| Why needed?.....   | 9  |
| Selenium Performance Testing with JMeter and Selenium Grid ..... | 16 |
| References .....   | 23 |

## JMeter – Selenium WebDriver Integration

**Goal:** Running Selenium tests in a way to re-use already automated (Java) Selenium scenarios instead of re-writing JS-scripts for [WebDriver Sampler](#), with the help of JUnit Request Sampler.

### JUnit Request Sampler

The benefit of Selenium for functional testing using JUnit Framework, provides the ability to utilize the same scripts for Performance testing in JMeter. Which makes the Open-Source Integration of Functional to Performance test, that is Selenium Integration with JMeter.

The current implementation supports standard JUnit convention and extensions, like **oneTimeSetUp** and **oneTimeTearDown**. Other features can be added on request. The sampler works like the JavaSampler with some differences.

- Rather than use JMeter's test interface, it scans the jar files for classes extending JUnit's **TestCase** class. This means any class or subclass.
- JUnit test jar files are copied to **jmeter/lib/junit** instead of **jmeter/lib**
- JUnit sampler does not use name/value pairs for configuration. The sampler assumes **setUp** and **tearDown** will configure the test correctly.

Note: **setUp** and **tearDown** methods must be declared **public access Modifier**, so that JMeter can use it.

- The sampler measures the elapsed time only for the test method and does not include **setUp** and **tearDown**.
- Each time the test method is called, JMeter will pass the result to the listeners.
- Support for **oneTimeSetUp** and **oneTimeTearDown** is done as a method. Since JMeter is multi-threaded, we cannot call **oneTimeSetUp/oneTimeTearDown** the same way maven does it.
- The sampler reports unexpected exceptions as errors.

## Step-by-step.

1. Write your JUnit test and jar the classes
2. Copy and paste the jar files into **jmeter/lib/junit** directory
3. Download Selenium Standalone Java client libraries and place it in the path "C:\JMETER\apache-jmeter-5.4.1\lib"
4. Start JMeter
5. Select **Test Plan**
6. Right click *Add* → *Thread Group*
7. Select **Thread Group**
8. Right click *Add* → *Sampler* → *JUnit Request*
9. Enter **my unit test** in the name
10. Enter the package of your JUnit test
11. Select the class you want to test
12. Select a method to test
13. Enter **test successful** in success message
14. Enter **1000** in success code
15. Enter **test failed** in failure message
16. Enter **0001** in failure code
17. Select the Thread Group
18. Right click *Add* → *Listener* → *View Results Tree*

One benefit of the JUnit sampler is it allows the user to select any method from a variety of unit tests to create a test plan. This should reduce the amount of code a user needs to write to create a variety of test scenarios. From a basic set of test methods, different sequences and tests can be created using JMeter's GUI.

## General Guidelines

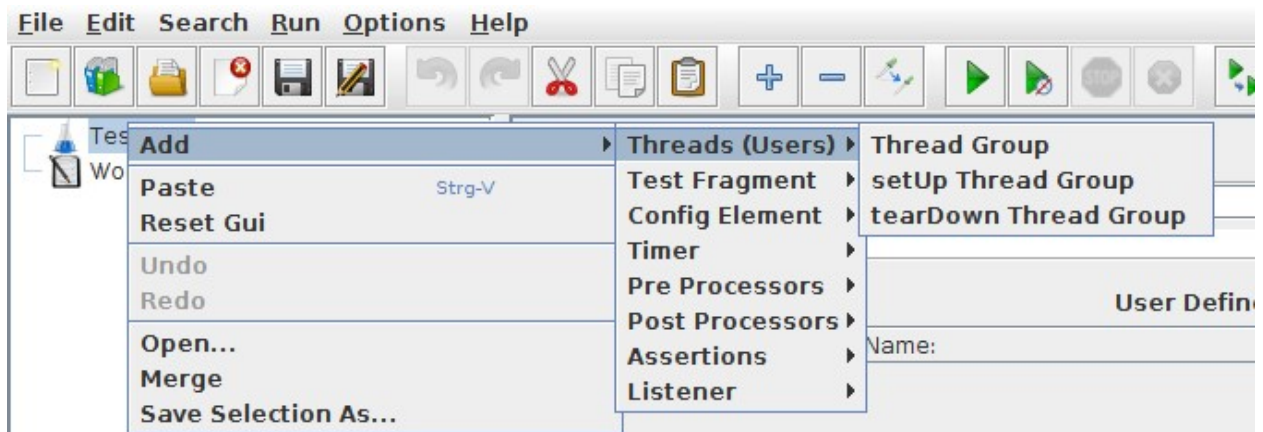
Here are some general guidelines for writing JUnit tests so they work well with JMeter. Since JMeter runs multi-threaded, it is important to keep certain things in mind.

- Write the **setUp** and **tearDown** methods so they are thread safe. This generally means avoid using static members.
- Make the test methods discrete units of work and not long sequences of actions. By keeping the test method to a discrete operation, it makes it easier to combine test methods to create new test plans.
- Avoid making test methods depend on each other. Since JMeter allows arbitrary sequencing of test methods, the runtime behavior is different than the default JUnit behavior.
- If a test method is configurable, be careful about where the properties are stored. Reading the properties from the Jar file is recommended.
- If you select a class and no methods show up, it means the sampler had a problem creating an instance of the test class. The best way to debug this is to add some **System.out** to your class constructor and see what is happening.
  - **NOTE:** Your test class should extend TestCase or SeleneseTestCase to allow JMeter pick up this test plan, test case's name **should start with "test"**).
  - **NOTE:** By default, SeleneseTestCase extends JUnit 3.x TestCase, also SeleneseTestCase expects external Selenium server to be running.
  -

## Step-by-Step-GUI

1. Start JMeter
2. Select **Test Plan**

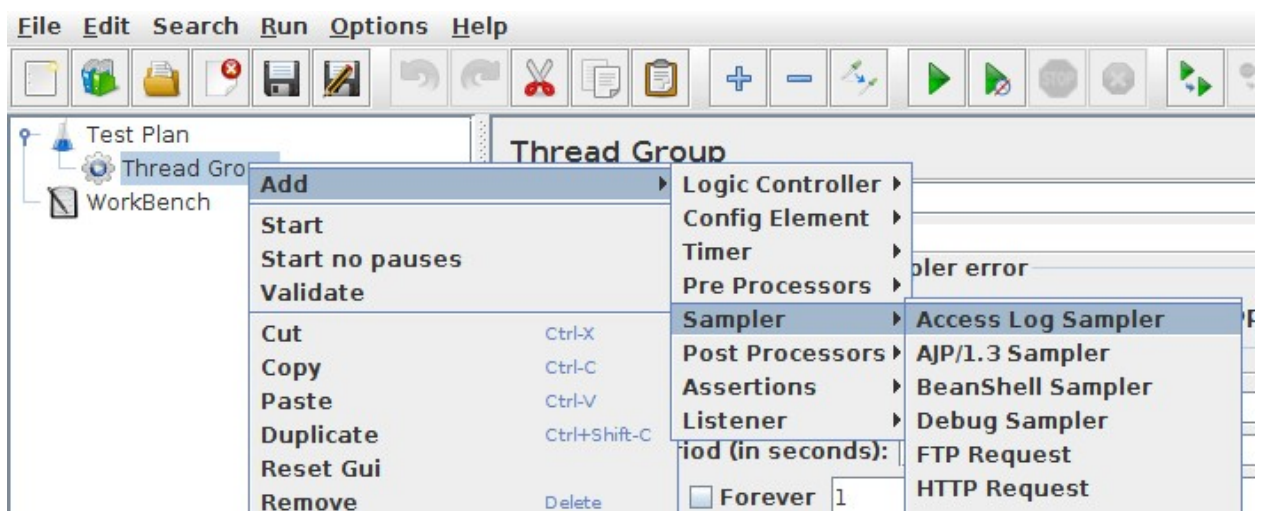
3. Right click *Add* → *Threads* (*Users*) → *Thread* *Group*



Add Thread Group

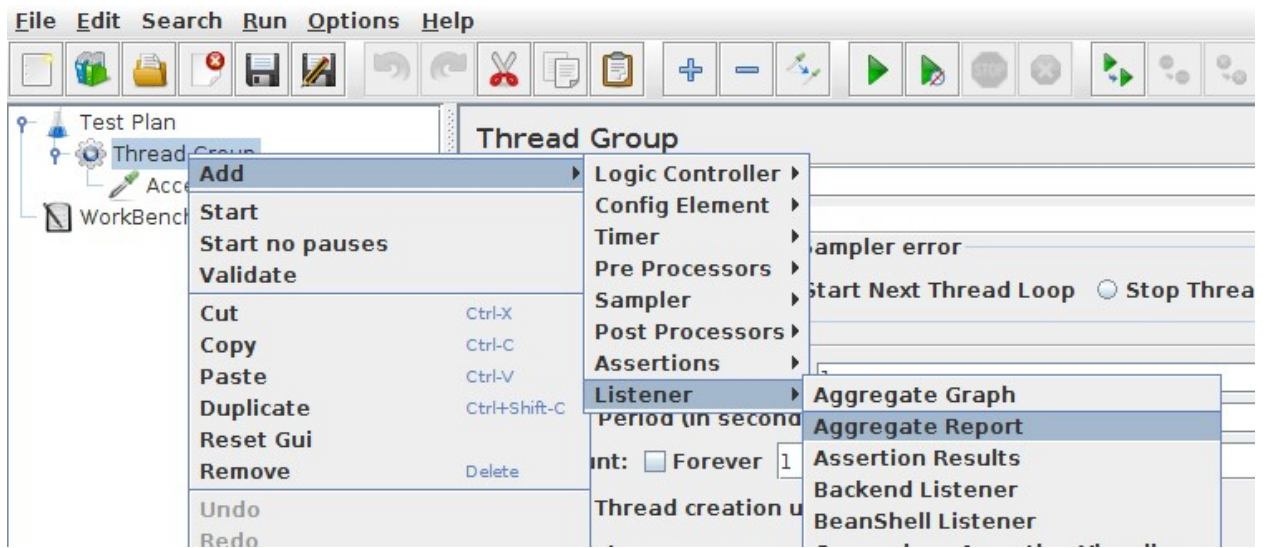
4. Select **Thread Group**

5. Right click *Add* → *Sampler* → *Access* *Log* *Sampler*



Add Access Log Sampler

6. Right click on **Thread Group** *Add* → *Listener* → *Aggregate* *Report*



Add Aggregate Report

7. Select the **Access Log** **Sampler**

 A screenshot of the 'Access Log Sampler' configuration dialog box. It has several sections:
 

- Name:** Access Log Sampler
- Comments:** (empty text area)
- Default Test Values:**
  - Protocol:** http
  - Server:** jmeter.apache.org
  - Port:** (empty)
  - Parse Images:** False
- Plugin Classes:**
  - Parser:** org.apache.jmeter.protocol.http.util.accesslog.TCLogParser
  - Filter (Optional):** org.apache.jmeter.protocol.http.util.accesslog.SessionFilter
- Log File Location:**
  - Log File:** (empty text field)
  - Browse...** button

Access Log Sampler

8. Enter the IP address or hostname in **Server**
9. Enter the port in **Port**
10. If you want to download the images, set **Parse images** to **true**.

11. Select a file for Log File Location

**Access Log Sampler**

Name: Access Log Sampler

Comments:

**Default Test Values**

Protocol: http

Server: 192.168.0.1

Port: 8080

Parse Images: False

**Plugin Classes**

Parser: org.apache.jmeter.protocol.http.util.accesslog.TCLogParser

Filter (Optional): Undefined

**Log File Location**

Log File: /tmp/access\_log.txt Browse...

Filled in Access Log Sampler

12. Select Aggregate Report

**Aggregate Report**

Name: Aggregate Report

Comments:

**Write results to file / Read from file**

Filename results.jtl Browse... Log/Disp

13. Enter **results.jtl** for filename
- | Label | # Sa... | Average | Median | 90% Li... | 95% Li... | 99 |
|-------|---------|---------|--------|-----------|-----------|----|
|-------|---------|---------|--------|-----------|-----------|----|
- Aggregate Report with filename

At this point, the test plan is ready. Start the test with **Ctrl + R** or from the menu *Start* → *Run*.

## JDBC Sampler

### MySQL Database and JMeter - How to Test Your Connection

When testing your APIs, web service or other system parts, you might need to record or retrieve data from a database. The purpose of this interaction is to check the correct record of specific data in the DB or to prepare test data for the tests by adding specific records to the database. This article will show you how to check, update, and add entries to your database by using [Apache JMeter™](#). Based on these examples, each tester can then perform the appropriate interaction with the database, during their further testing.



In this blog post we will use **JMeter 3.2**, the database [MySQL 5.7.18 Community Edition](#) (which is free and can be installed on your PC) and **Java8**.

Before you start working with a database by using [JMeter](#), you need to do the following:

- Make sure there is a user who has permission to connect and perform common actions CRUD in the database
- Make sure the database is available for remote or local access
- Install and choose the right properties for JMeter and the Java Development Kit

Download:

<https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.27>

<https://jmeter.apache.org/usermanual/build-db-test-plan.html>

## WebDriver Sampler

<https://jmeter-plugins.org/wiki/WebDriverSampler/> → Documentation.

JMeter can be integrated with Selenium via WebDriver Plugin support. Download the plugin and copy the jars in your "lib" folder and "ext" folder. The WebDriver sampler comes with config elements plugins for IE, Chrome, Mozilla and other browsers so that they can be invoked via selenium code written in your sampler. Set the driver path in the config element so that it picks the browser driver correctly and establish connected between your code and browser.

- You can configure other properties too in config element.
- Add a listener to your Test Plan having Webdriver sampler so that you get to know the time taken by the user to perform those actions.
- Keep your log viewer on so that errors can be visualized while running you tests.

# Why needed?

With the advancement of technology, HTML5, JS and CSS improvements, more and more logic and behaviour have been pushed down to the client. Things that add to the overall browser execution time may include:

1. Client-side Javascript execution – eg. AJAX, JS templates
2. CSS transforms – eg. 3D matrix transforms, animations
3. 3rd party plugins – eg. Facebook like, Double click ads, site analytics, etc

All these things add to the overall browser execution time,

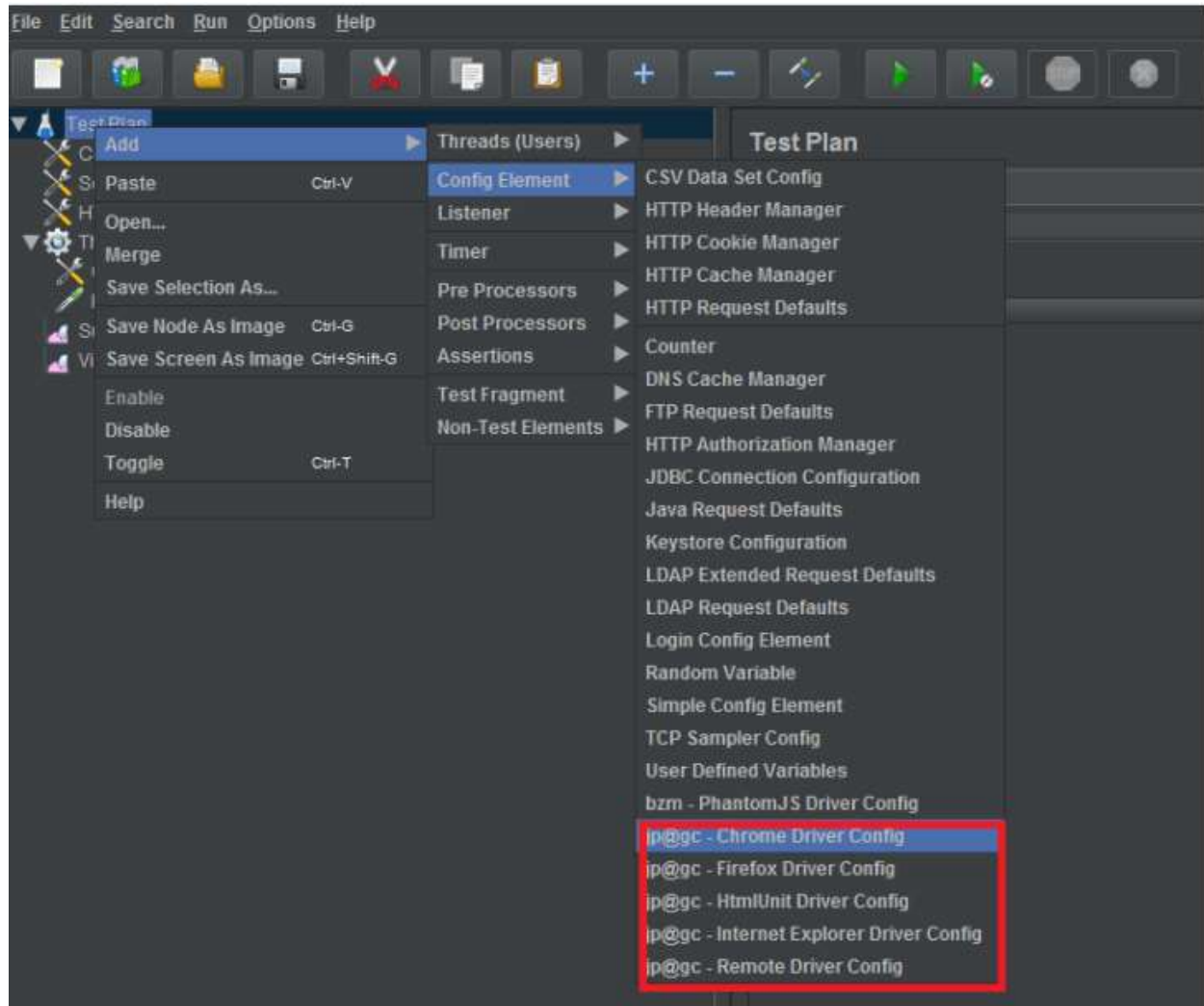
This adds to the overall perceived performance of website/webapp, but this metric is not available in JMeter. JMeter is not a real browser so we can't measure the user experience at client side, like page rendering/load time.

Performance mix is the practice to test application user experience while performing a load test. Web Driver Sampler automates the execution and collection of Performance metrics on the Browser (client-side). JMeter Load Test will put enough load on your system while the JMeter WebDriver plan will allow you to get the user experienced response times including page rendering.

## Pre-requisite:

1. JMeter must be installed on your system if not then refer [here](#).
2. Install the [Webdriver Set](#) plugin using the [JMeter Plugins Manager](#).

Write your WebDriver script as usual, then add “Thread Group” to your “Test Plan.”



Add below JMeter elements to test plan.

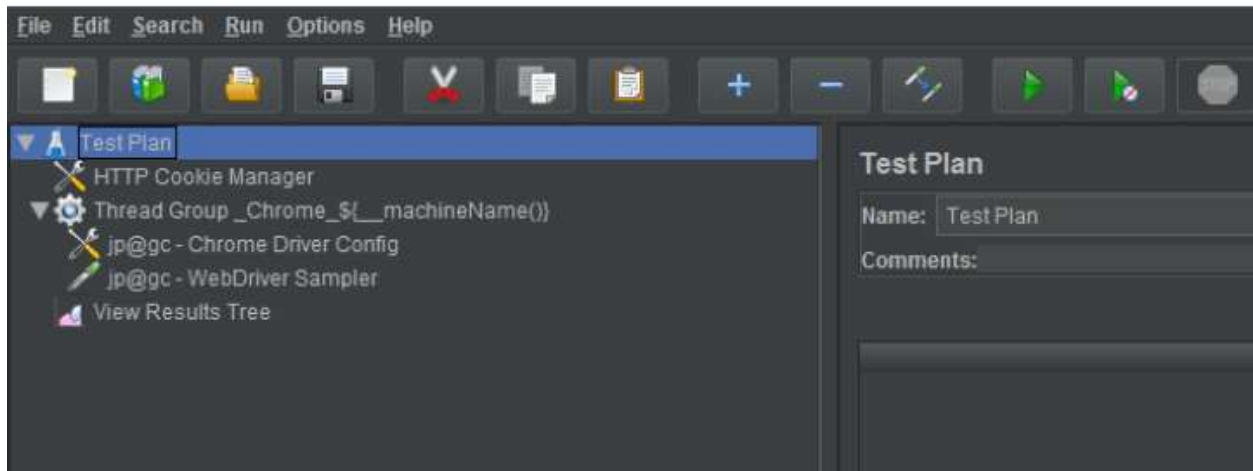
Config Element -> HTTP Cookie Manager

Config Element -> jp@gc – Chrome Driver Config

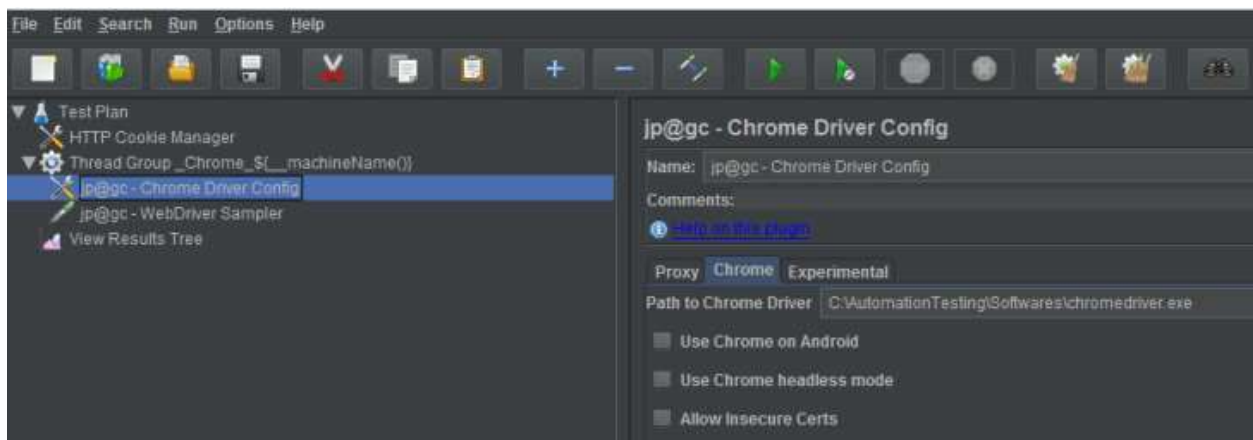
Sampler -> jp@gc – Web Driver Sampler

Listener -> View Results Tree.

The result is as follows:



Add chrome driver path to the “***Path to Chrome Driver***” .



Add below code to WebDriver and modify as per your business scenario :

```
var pkg = JavaImporter(org.openqa.selenium,
org.openqa.selenium.support.ui,org.openqa.selenium.support.events.EventFiringWebDriver);

var wait = new pkg.WebDriverWait(WDS.browser, 150)// WebDriver wait

WDS.sampleResult.sampleStart(); //captures sampler's start time

WDS.sampleResult.getLatency();
```

```
WDS.log.info("Sample started");

// Launch website specified in 'http://newtours.demoaut.com/'

WDS.browser.get('http://newtours.demoaut.com/');

WDS.log.info("Sample ended - navigated to
http://newtours.demoaut.com/");

//Enter user name

wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg
.By.name('userName')));

var txtUserName =
WDS.browser.findElement(pkg.By.name('userName')); //saves
username field into txtUserName

txtUserName.sendKeys(['mercury']); //enter 'mercury' in user
name

WDS.log.info("Enter UserName");

//Enter password

wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg
.By.name('password')));

var txtPassword =
WDS.browser.findElement(pkg.By.name('password')); //saves
password field into txtPassword
```

```
txtPassword.sendKeys(['mercury']); //enter 'mercury' in
password

WDS.log.info("Enter password");

//Click Sign in button

wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg
.By.xpath('//input[@name=\'login\']')));

var btnLogin =
WDS.browser.findElement(pkg.By.xpath('//input[@name=\'login\']
')); //saves login field into btnLogin

btnLogin.click();

//Click sign off link

wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg
.By.xpath('//a[@href=\'mercurysignoff.php\']')));

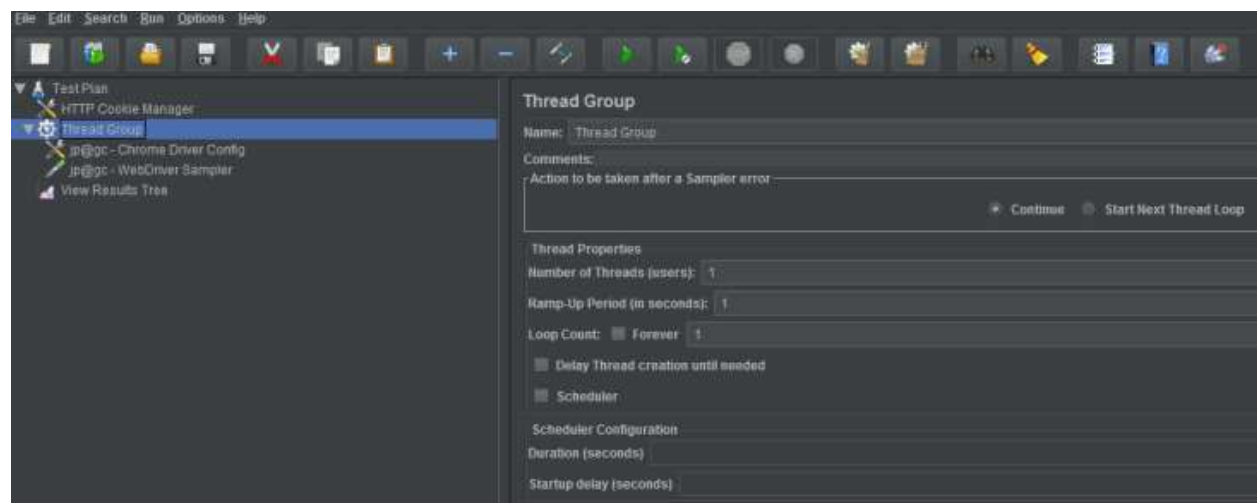
var lnkSignOff =
WDS.browser.findElement(pkg.By.xpath('//a[@href=\'mercurysigno
ff.php\']')); //saves sign off field into lnkSignOff

lnkSignOff.click();
```

```
WDS.sampleResult.sampleEnd();
```

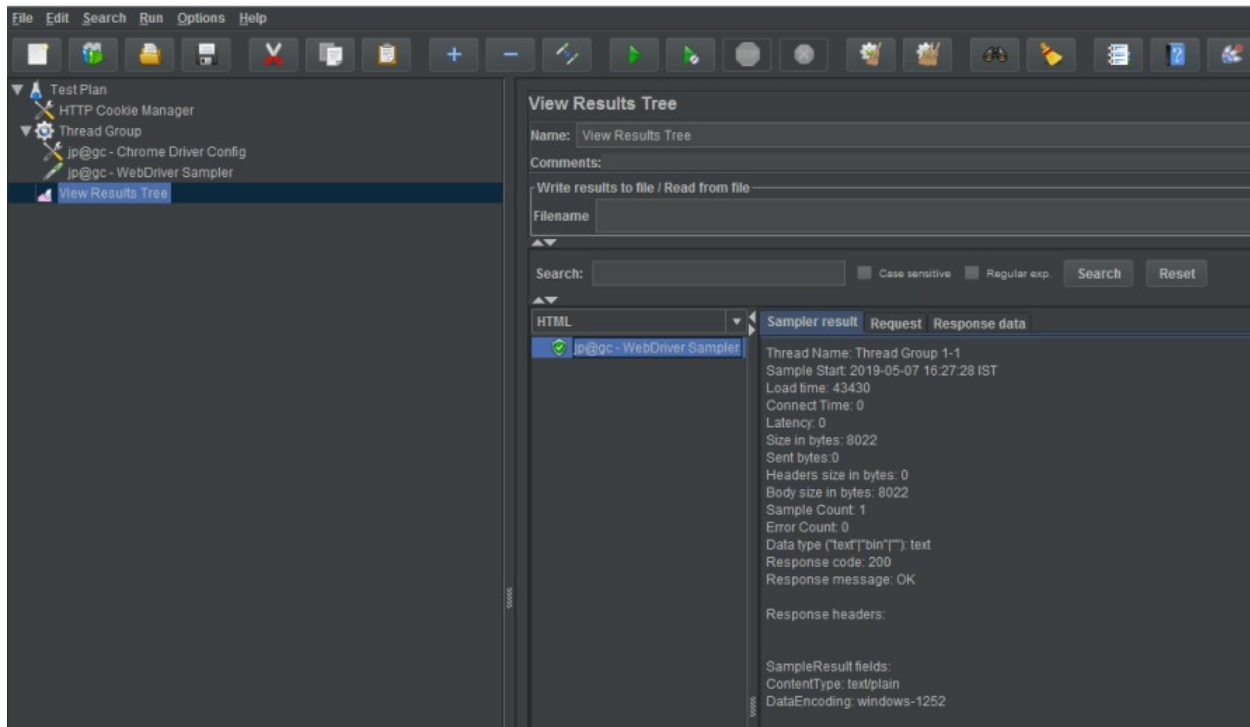
```
jp@gc - WebDriver Sampler
Name: jp@gc - WebDriver Sampler
Comments:
Parameters:
Script Language: javascript
Script (see below for variables that are defined)
1 var pkg = JavaImporter(org.openqa.selenium, org.openqa.selenium.support.ui, org.openqa.selenium.support.events.EventFiringWebDriver);
2 var wait = new pkg.WebDriverWait(WDS.browser, 150); // WebDriver wait
3
4 WDS.sampleResult.sampleStart(); //captures sampler's start time
5 WDS.sampleResult.getLatency();
6 WDS.log.info("Sample started");
7
8 // Launch website specified in 'http://newtours.demoaut.com/'
9 WDS.browser.get('http://newtours.demoaut.com/');
10 WDS.log.info("Sample ended - navigated to http://newtours.demoaut.com/");
11
12 //Enter user name
13 wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg.By.name('userName')));
14 var txtUserName = WDS.browser.findElement(pkg.By.name('userName')); //saves username field into txtUserName
15 txtUserName.sendKeys(['mercury']); //enter 'mercury' in user name
16 WDS.log.info("Enter UserName");
17
18 //Enter password
19 wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg.By.name('password')));
20 var txtPassword = WDS.browser.findElement(pkg.By.name('password')); //saves password field into txtPassword
21 txtPassword.sendKeys(['mercury']); //enter 'mercury' in password
22 WDS.log.info("Enter password");
23
24 //Click Sign in button
25 wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg.By.xpath('//input[@name=\'login\']')));
26 var btnLogin = WDS.browser.findElement(pkg.By.xpath('//input[@name=\'login\']')); //saves login field into btnLogin
27 btnLogin.click();
28
29 //Click sign off link
30 wait.until(pkg.ExpectedConditions.presenceOfElementLocated(pkg.By.xpath('//a[@href=\'mercurysignoff.php\']')));
31 var lnkSignOff = WDS.browser.findElement(pkg.By.xpath('//a[@href=\'mercurysignoff.php\']')); //saves sign off field into lnkSignOff
32 lnkSignOff.click();
33
34
35
36 WDS.sampleResult.sampleEnd();
```

Now, try to start your test. Whatever you do, DO NOT change the “Thread Group” values. They must all be set to 1.



You should see the new Chrome window that will open the website. login to mercury website . After the test has started, open View Results Tree to confirm there are no

errors. If the Response Code is “200” and the Response Message is “OK,” the test was run successful. If not, check the WebDriver script for errors.





## Selenium Performance Testing with JMeter and Selenium Grid

In this post, we will complete the Selenium Performance testing scenario using JMeter and Selenium Grid.

1- Install Java 7 or later If necessary

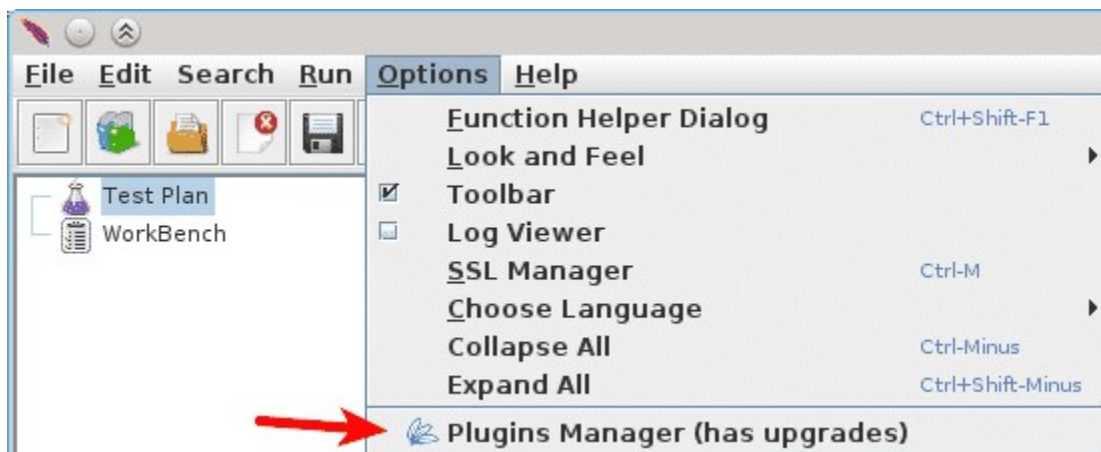
<https://java.com/tr/download/>

2- Download latest Jmeter version 3.0 or higher.

[http://jmeter.apache.org/download\\_jmeter.cgi](http://jmeter.apache.org/download_jmeter.cgi)

3- Download Jmeter PluginsManager JAR file and put it to Jmeter's lib/ext directory. Then, start JMeter and go to "Options" menu to access the Plugins Manager. Open Jmeter Plugins Manager and Install "Selenium/Webdriver Support" plugin. Restart Jmeter to take changes for Selenium.

<https://jmeter-plugins.org/wiki/PluginsManager/>



Now you can use the following libraries with numerous script languages !!

4- Download latest stable version of Selenium Grid (selenium-server-standalone.jar)

<http://selenium-release.storage.googleapis.com/index.html>

**NOTE:** If you are using Selenium Grid with your own PC, browser windows will be popout at your PC and will be harder. You should test it on your own PC but you will need an test automation machine. I recommend that, you should use a headless Linux ubuntu server for Selenium Grid execution. The following command must be working successfully.

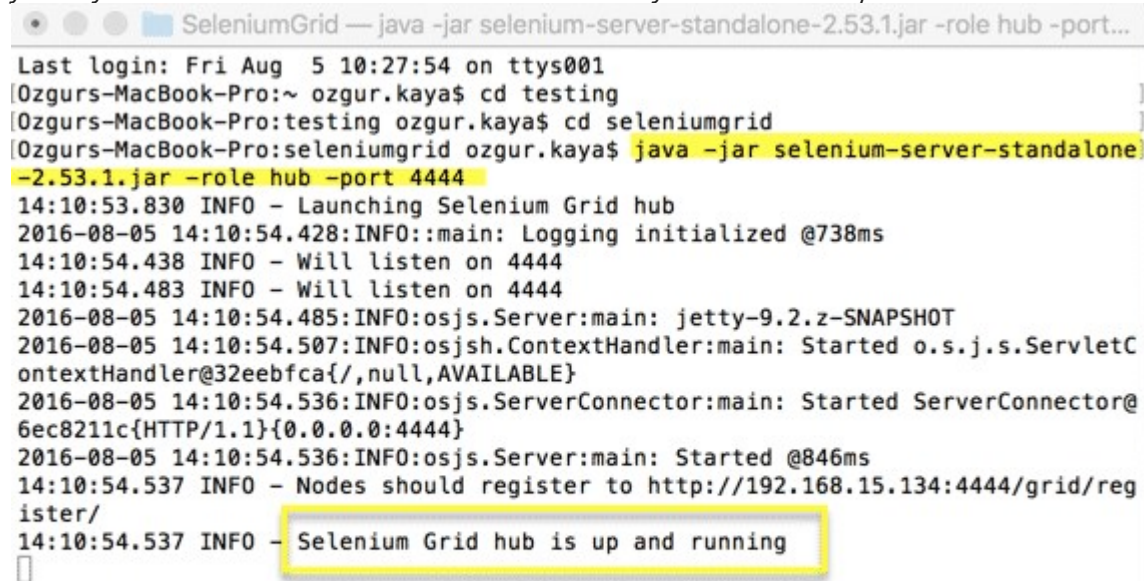
5- Download your browser driver. We will use Chrome driver in this example.

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

## 6- Run Selenium Grid Hub

Open the terminal windows or command prompt. Go to the JAR file location and Execute the following command in the first window. This will start the selenium hub.

```
java -jar selenium-server-standalone-2.53.1.jar -role hub -port 4444
```



```

Last login: Fri Aug 5 10:27:54 on ttys001
Ozgurs-MacBook-Pro:~ ozgur.kaya$ cd testing
Ozgurs-MacBook-Pro:testing ozgur.kaya$ cd seleniumgrid
Ozgurs-MacBook-Pro:seleniumgrid ozgur.kaya$ java -jar selenium-server-standalone-2.53.1.jar -role hub -port 4444
14:10:53.830 INFO - Launching Selenium Grid hub
2016-08-05 14:10:54.428:INFO::main: Logging initialized @738ms
14:10:54.438 INFO - Will listen on 4444
14:10:54.483 INFO - Will listen on 4444
2016-08-05 14:10:54.485:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2016-08-05 14:10:54.507:INFO:osjs.ContextHandler:main: Started o.s.j.s.ServletContextHandler@32eebfca{/,null,AVAILABLE}
2016-08-05 14:10:54.536:INFO:osjs.ServerConnector:main: Started ServerConnector@6ec8211c{HTTP/1.1}{0.0.0.0:4444}
2016-08-05 14:10:54.536:INFO:osjs.Server:main: Started @846ms
14:10:54.537 INFO - Nodes should register to http://192.168.15.134:4444/grid/register/
14:10:54.537 INFO - Selenium Grid hub is up and running

```

## 7- Register node to Selenium Grid Hub

Go to the JAR file location and Execute the following command in the second new terminal window. This will start and configure the nodes which you will use it for performance testing. If you need to do a performance testing with 100 concurrent users, you must set maxInstances value to 100. The following code only registers the chrome browsers. You should configure it with mixed browser settings as well.

```

java -jar selenium-server-standalone-2.53.1.jar -role node -hub
http://localhost:4444/grid/register -maxSession 100 -browser
browserName="chrome",version=ANY,platform=WINDOWS,maxInstances=50 -
Dwebdriver.chrome.driver=path/to/the/chromedriver/chromedriver_ForMac

```

```
SeleniumGrid — -bash — 80x24
Last login: Fri Aug  5 14:09:52 on ttys000
Ozgurs-MacBook-Pro:~ ozgur.kaya$ cd testing
Ozgurs-MacBook-Pro:testing ozgur.kaya$ cd seleniumgrid
Ozgurs-MacBook-Pro:seleniumgrid ozgur.kaya$ java -jar selenium-server-standalone
-2.53.1.jar -role node -hub http://localhost:4444/grid/register -maxSession 100
-browser browserName="chrome",version=ANY,platform=WINDOWS,maxInstances=50 -Dweb
driver.chrome.driver=path/to/the/chromedriver/chromedriver_ForMac

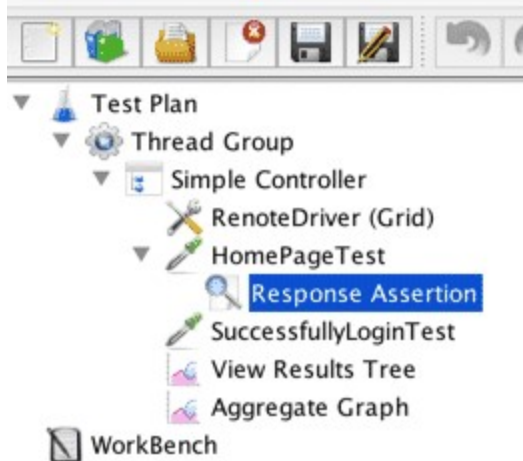
14:13:07.668 INFO - Selenium Grid node is up and ready to register to the hub
14:13:07.688 INFO - Starting auto registration thread. Will try to register ever
y 5000 ms.
14:13:07.688 INFO - Registering the node to the hub: http://localhost:4444/grid/
register
14:13:07.712 INFO - The node is registered to the hub and ready to use
```

8- Check your Selenium Grid is up and running correctly



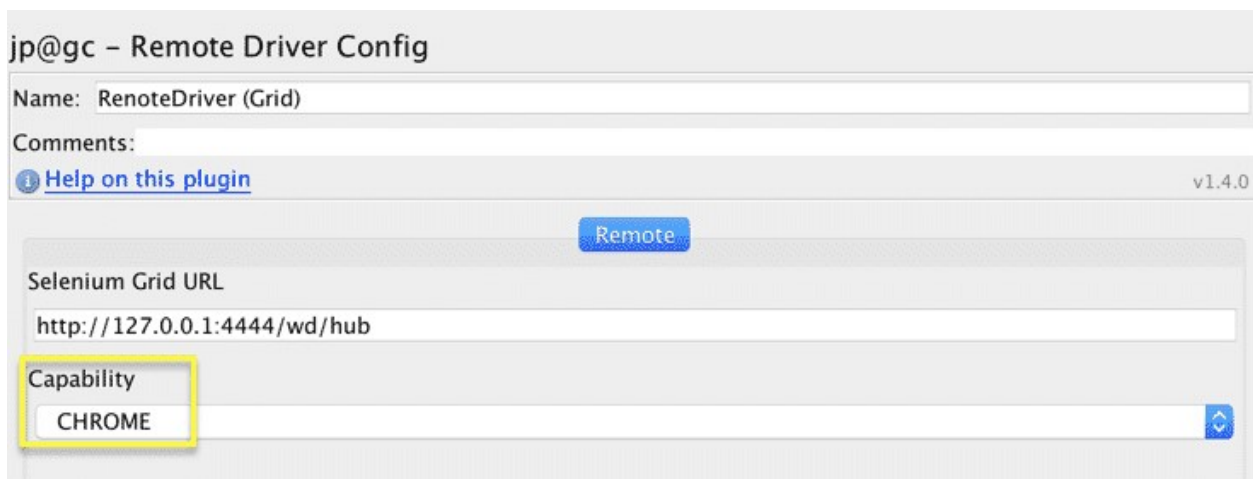
9- Run Jmeter from bin/jmeter directory. Add the following elements to your project.

- Right click to Test Plan and Add Threads (Users->Thread Group)
- Right click to Thread Group and Add Config Element -> Simple Controller
- Right click to Simple Controller and Add Logic Controller -> jp@gc-Remote Driver Config
- Right click to Simple Controller and Sampler -> jp@gc WebDriver Sample
- Right click to Simple Controller and Sampler -> jp@gc WebDriver Sample
- Right click to Simple Controller and Listeners -> View Results Tree
- Right click to Simple Controller and Listeners -> Aggregate Graph



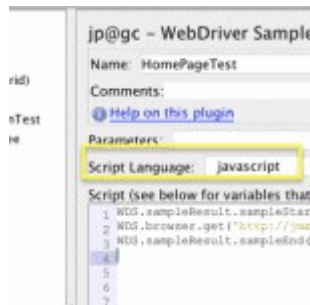
10- Click Remote Driver Config and write your selenium grid URL. You should change 127.0.0.1 to your real existing selenium grid server IP/url. Change capability to Chrome. If you want to use different browser, you must reRegister node to Grid Hub with different settings.

<http://127.0.0.1:4444/wd/hub>



11- Click WebDriver Sample on the left menu. You should write your test scenarios with a lot of scripting languages to here. You should rename your webdriver samples for understandable test scenarios.

There are some useful examples @this link: (Reference: <https://jmeter-plugins.org/wiki/WebDriverSampler/>)

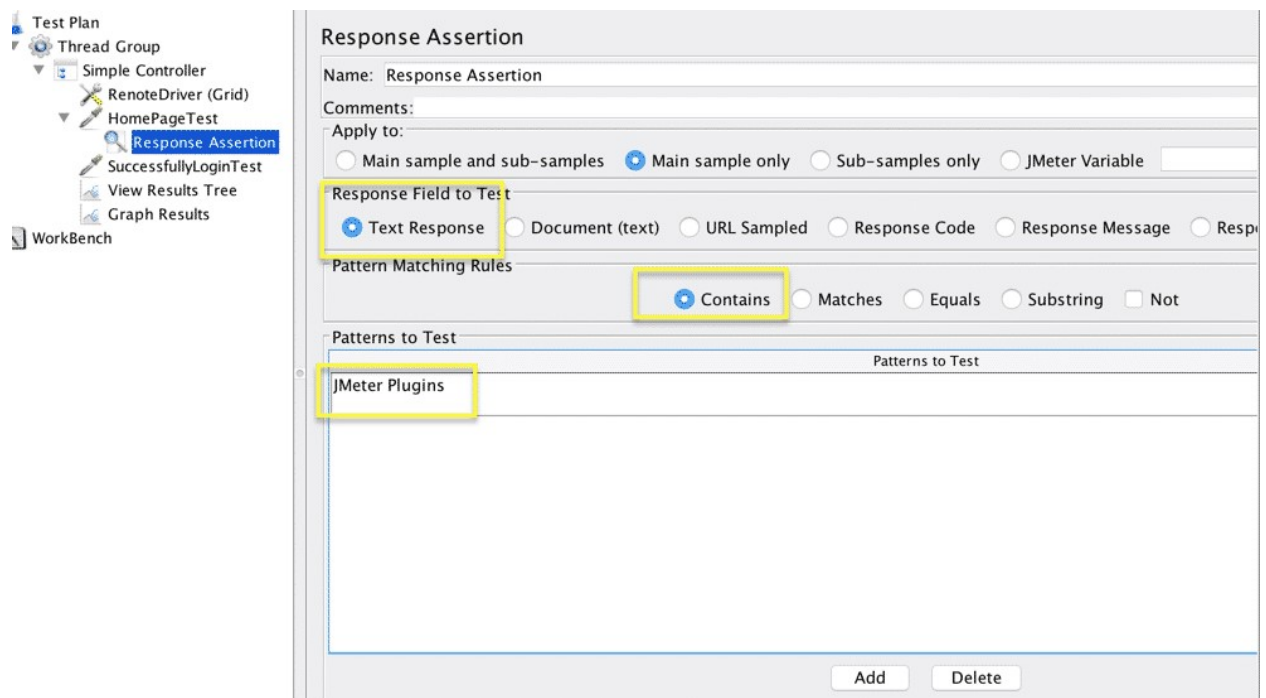


## 12- Write test asserts

Right click to any Webdriver Sampler and Add Assertions→ Response Assertion.

Right click to any Webdriver Sampler and Add Assertions→ Duration Assertion.

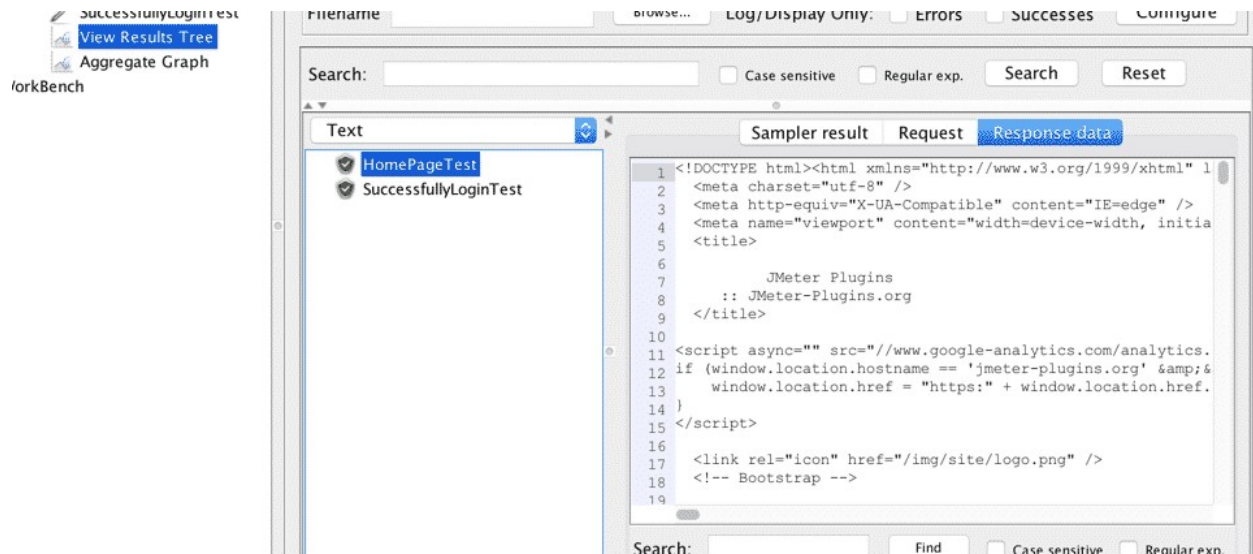
Right click to any Webdriver Sampler and Add Assertions→ Size Assertion.....



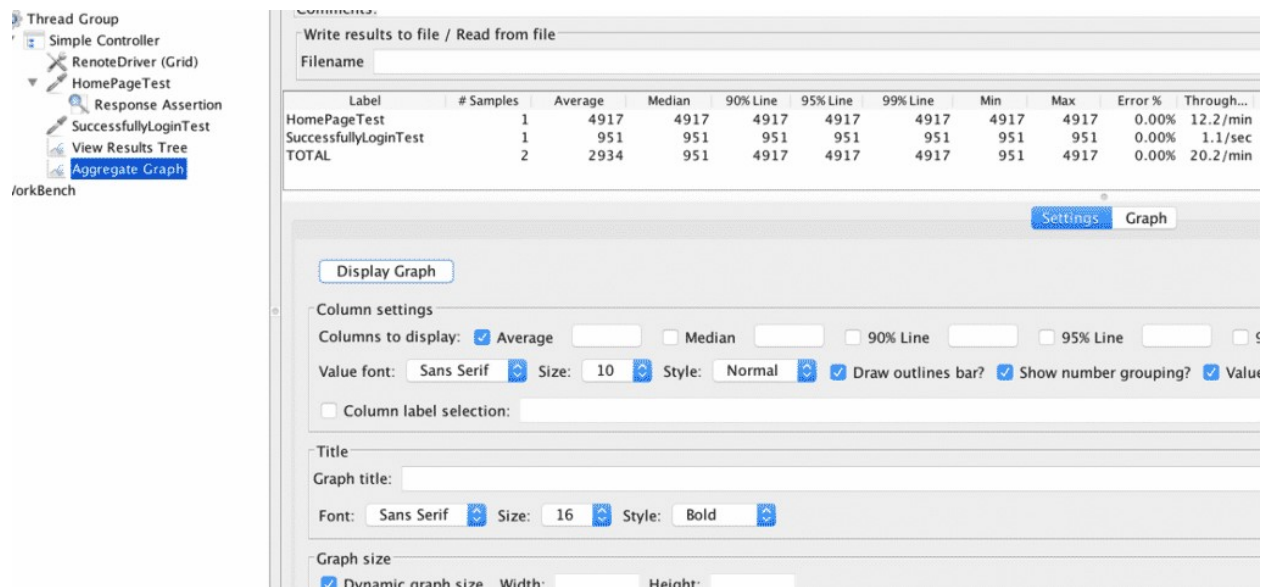
**Note:** Any response codes except 2xx or 3xx will fail automatically. For example 500 Server Error response code will be automatically fail by Jmeter.

## 13- View request and response details.(View Results Tree)

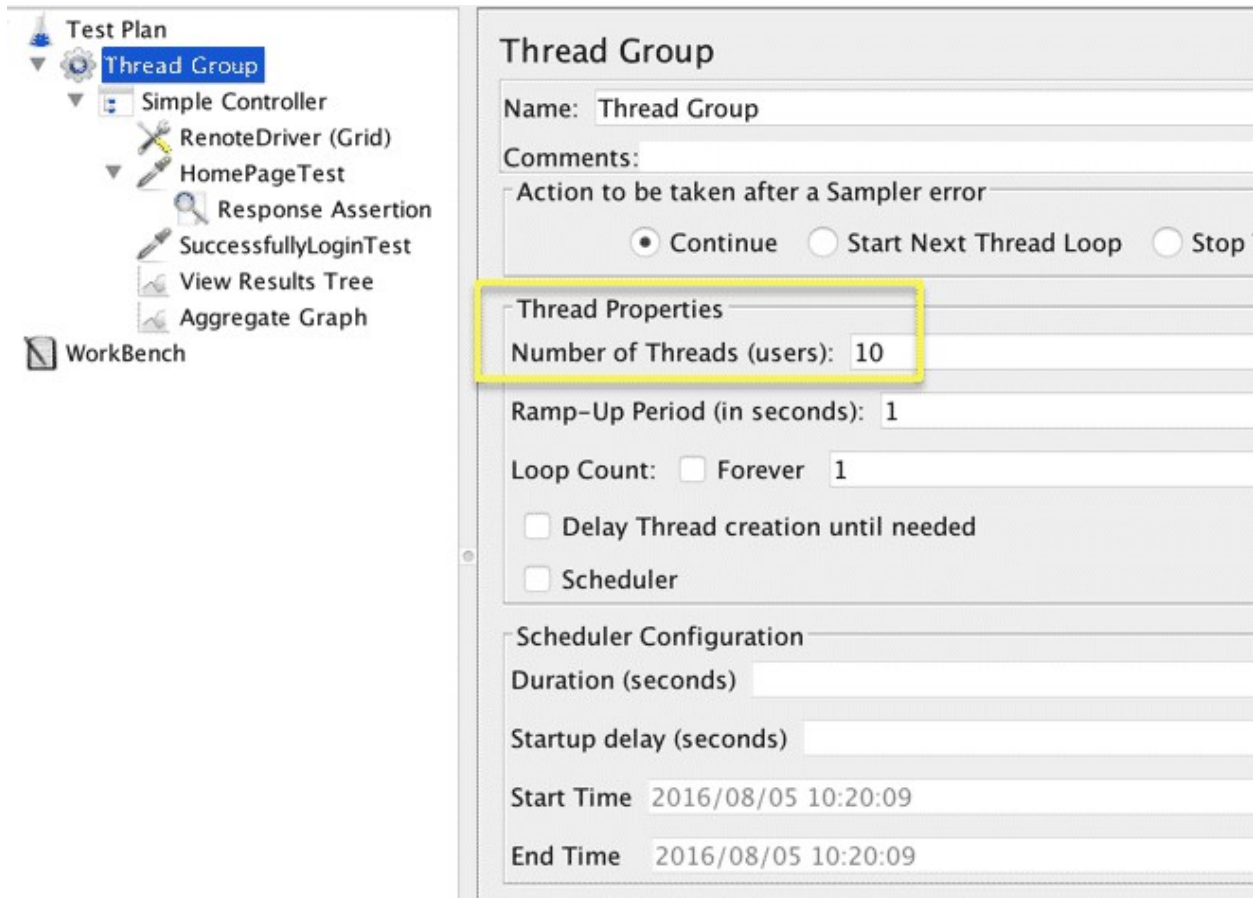




#### 14- View Overall test results listed or graphical. (Aggregate Graph)



15- Run a Load Test with concurrent 10 users(10 Thread). And You should increase this value step by step. Set Loop count for longer testing.



The screenshot shows the JMeter configuration interface. On the left, the Test Plan tree is expanded, showing a Thread Group containing a Simple Controller, RemoteDriver (Grid), HomePageTest, Response Assertion, SuccessfullyLoginTest, View Results Tree, and Aggregate Graph. The Thread Group is selected. On the right, the Thread Group configuration panel is displayed. The Name is 'Thread Group'. The Number of Threads (users) is set to 10. The Ramp-Up Period (in seconds) is 1. The Loop Count is set to 1. The Delay Thread creation until needed checkbox is unchecked. The Scheduler checkbox is unchecked. The Scheduler Configuration section shows Duration (seconds) and Startup delay (seconds) as empty fields. The Start Time is 2016/08/05 10:20:09 and the End Time is 2016/08/05 10:20:09.

**Thread Group**

Name: Thread Group

Comments:

Action to be taken after a Sampler error

☒ Continue ☐ Start Next Thread Loop ☐ Stop

**Thread Properties**

Number of Threads (users): 10

Ramp-Up Period (in seconds): 1

Loop Count: ☐ Forever 1

☐ Delay Thread creation until needed

☐ Scheduler

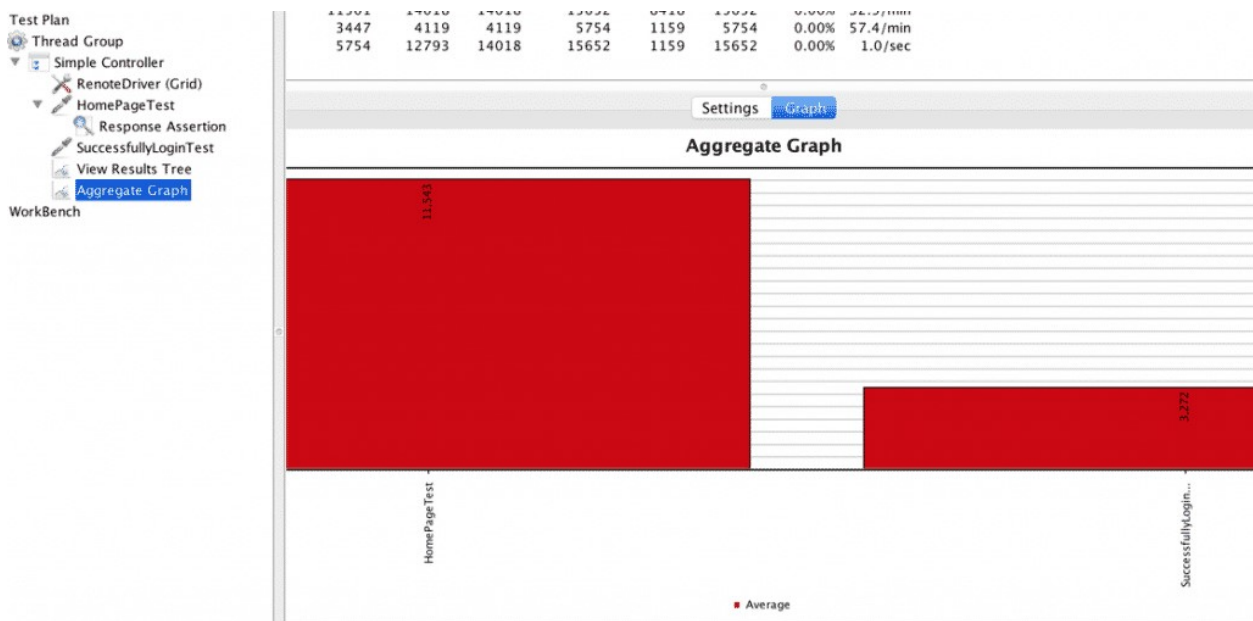
**Scheduler Configuration**

Duration (seconds)

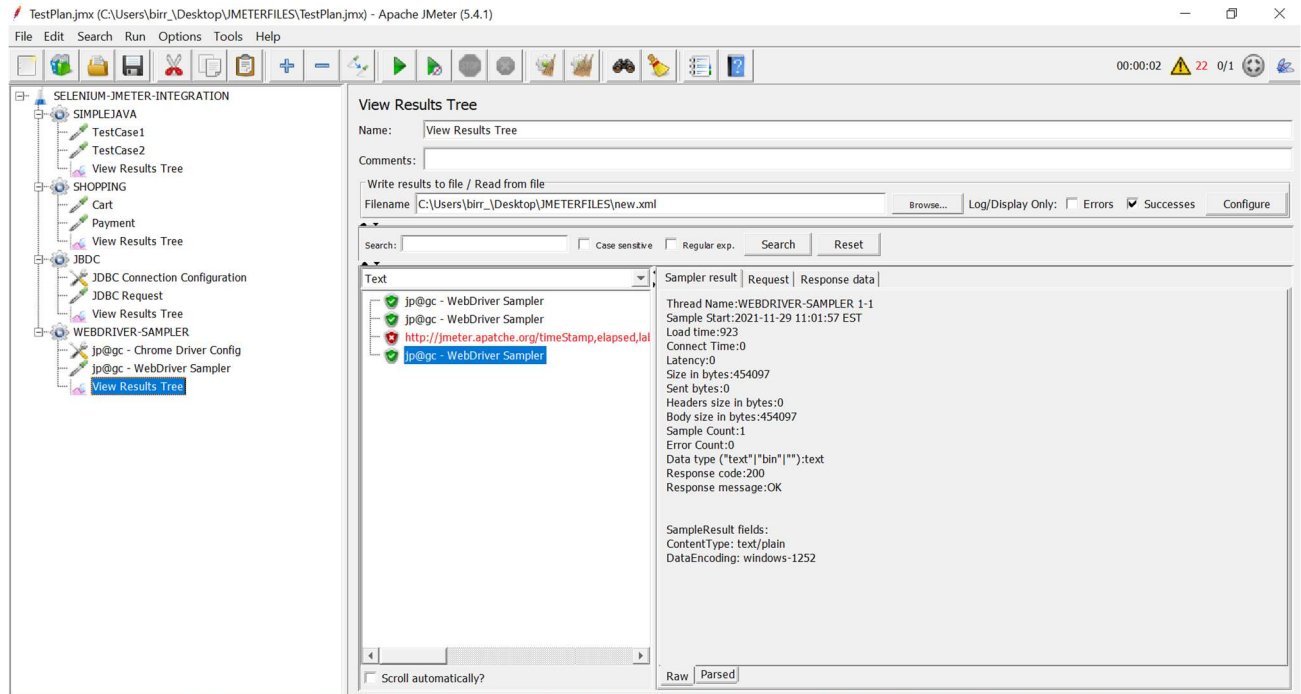
Startup delay (seconds)

Start Time 2016/08/05 10:20:09

End Time 2016/08/05 10:20:09



***Note:** Don't forget that you must disable or Configure for "Log/Display only Errors" View Results Tree if you don't use it. If you don't do that while running load tests with multiple users, You should get not enough memory or memory leaks problems.*



## References

1. [https://jmeter.apache.org/usermanual/junitsampler\\_tutorial.html](https://jmeter.apache.org/usermanual/junitsampler_tutorial.html)
2. <https://jmeter-plugins.org/wiki/WebDriverSampler/>
3. <https://www.selenium.dev/>