Selenium-Grid

*Note: We are currently working on this chapter. Presently we have introductory info here for people completely new to Selnium-Grid. Over the next few months we hope to provide useful examples and illustrations to thoroughly explain how to use Selenium-Grid.*

Quick Start

If you’re already experienced in Selenium test automation you may simply need a quick-start to get up and running. This chapter has much information geared to many skill levels, but may be too much if you’re looking for just a quick reference to quickly try things out. For a quick-start, refer to the Selenium-Grid articles in the [Selenium Wiki](http://code.google.com/p/selenium/wiki/Grid2).

What is Selenium-Grid?

Selenium-Grid allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution. It allows for running your tests in a *distributed test execution* environment.

When to Use It

Generally speaking, there’s two reasons why you might want to use Selenium-Grid.

* To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
* To reduce the time it takes for the test suite to complete a test pass.

Selenium-Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel. For example, if you have a suite of 100 tests, but you set up Selenium-Grid to support 4 different machines (VMs or separate physical machines) to run those tests, your test suite will complete in (roughly) one-fourth the time as it would if you ran your tests sequentially on a single machine. For large test suites, and long-running test suite such as those performing large amounts of data-validation, this can be a significant time-saver. Some test suites can take hours to run. Another reason to boost the time spent running the suite is to shorten the turnaround time for test results after developers check-in code for the AUT. Increasingly software teams practicing Agile software development want test feedback as immediately as possible as opposed to wait overnight for an overnight test pass.

Selenium-Grid is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time. For example, a ‘grid’ of virtual machines can be setup with each supporting a different browser that the application to be tested must support. So, machine 1 has Internet Explorer 8, machine 2, Internet Explorer 9, machine 3 the latest Chrome, and machine 4 the latest Firefox. When the test suite is run, Selenium-Grid receives each test-browser combination and assigns each test to run against its required browser.

In addition, one can have a grid of all the same browser, type and version. For instance, one could have a grid of 4 machines each running 3 instances of Firefox 12, allowing for a ‘server-farm’ (in a sense) of available Firefox instances. When the suite runs, each test is passed to Selenium-Grid which assigns the test to the next available Firefox instance. In this manner one gets test pass where conceivably 12 tests are all running at the same time in parallel, significantly reducing the time required to complete a test pass.

Selenium-Grid is very flexible. These two examples can be combined to allow multiple instances of each browser type and version. A configuration such as this would provide both, parallel execution for fast test pass completion and support for multiple browser types and versions simultaneously.

Selenium-Grid 2.0

Selenium-Grid 2.0 is the latest release as of the writing of this document (5/26/2012). It is quite different from version 1 of Selenium-Grid. In 2.0 Selenium-Grid was merged with the Selenium-RC server. Now, you only need to download a single .jar file to get the remote Selenium-RC-Server and Selenium-Grid all in one package.

How Selenium-Grid Works–With a Hub and Nodes

A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable. We’ve listed some examples in the following sections of this chapter.

The hub receives a test to be executed along with information on which browser and ‘platform’ (i.e. WINDOWS, LINUX, etc) where the test should be run. It ‘knows’ the configuration of each node that has been ‘registered’ to the hub. Using this information it selects an available node that has the requested browser-platform combination. Once a node has been selected, Selenium commands initiated by the test are sent to the hub, which passes them to the node assigned to that test. The node runs the browser, and executes the Selenium commands within that browser against the application under test.

Installation

Installation is simple. Download the Selenium-Server jar file from the [SeleniumHq website’s download page](http://seleniumhq.org/download/). You want the link under the section “Selenium-Server (formerly Selenium-RC)”.

Install it in a folder of your choice. You’ll need to be sure the java executable is on your execution path so you can run it from the command-line. If it does not run correcly, verify your system’s path variable includes the path to the java.exe.

Starting Selenium-Grid

Generally you would start a hub first since nodes depend on a hub. This is not abolutely necessary however, since nodes can recognize when a hub has been started and vice-versa. For learning purposes though, it would easier to start the hub first, otherwise you’ll see error messages that may not want to start off with your first time using Selenium-Grid.

**Starting a Hub**

To start a hub with default parameters, run the following command from a command-line shell. This will work on all the supported platforms, Windows Linux, or Mac OSX.

java -jar selenium-server-standalone-2.44.0.jar -role hub

This starts a hub using default parameter values. We’ll explain these parameters in folowing subsections. Note that you will likely have to change the version number in the jar filename depending on which version of the selenium-server you’re using.

**Starting a Node**

To start a node using default parameters, run the following command from a command-line.

java -jar selenium-server-standalone-2.44.0.jar -role node -hub http://localhost:4444/grid/register

This assumes the hub has been started above using default parameters. The default port the hub uses to listen for new requests is port 4444. This is why port 4444 was used in the URL for locating the hub. Also the use of ‘localhost’ assumes your node is running on the same machine as your hub. For getting started this is probably easiest. If running the hub and node on separate machines, simply replace ‘localhost’ with the hostname of the remote machine running the hub.

WARNING: Be sure to turn off the firewalls on the machine running your hub and nodes. Otherwise you may get connection errors.

Configuring Selenium-Grid

**Default Configuration**

**JSON Configuration File**

**Configuring Via the Command-Line Options**

Hub Configuration

To run the hub using the default options simply specify -role hub to the Selenim-Server

java -jar selenium-server-standalone-2.44.0.jar -hub

You should see the following logging output.

Jul 19, **2012** 10:46:21 AM org.openqa.grid.selenium.GridLauncher main

INFO: Launching a selenium grid server

2012-07-19 10:46:25.082:INFO:osjs.Server:jetty-7.x.y-SNAPSHOT

2012-07-19 10:46:25.151:INFO:osjsh.ContextHandler:started o.s.j.s.ServletContextHandler{/,null}

2012-07-19 10:46:25.185:INFO:osjs.AbstractConnector:Started SocketConnector@0.0.0.0:4444

**Specifying the Port**

The default port used by the hub is 4444. The port being referred to here, is the TCP/IP port used when the ‘client’, that is, the automated tests connect to the Selenium-Grid hub. If another application on your computer is already using this port, or if, you already have a Selenium-Server started, you’ll see the following message in the log output.

10:56:35.490 WARN - Failed to start: SocketListener0@0.0.0.0:4444

Exception in thread "main" java.net.BindException: Selenium is already running on port 4444. Or some other service is.

If this occurs you can either shutdown the other process that is using port 4444, or you can tell Selenium-Grid to use a different port for its hub. Use the -port option for changing the port used by the hub.

java -jar selenium-server-standalone-2.44.0.jar -role hub -port 4441

This will work even if another hub is already running on the same machine, that is, as long as they’re both not using port 4441.

You may, however, want to see what process is using port 4444 so you can allow the hub to use the default. To see the ports used by all running programs on your machine use the command.

netstat -a

This should work on all supported systems, Unix/Linux, MacOs, and Windows although additional options beyond -a may be required. Basically you need to display the process ID along with the port. In Unix you may ‘grep’ the output (use a pipe) from the port number to only display those records you’re concerned with.

Node Configuration

Timing Parameters

Customizing the Grid

**Adding custom servlets at the hub and/or node**

The Grid lets you define your own servlets and then plug them into either the Hub (or) into the node. This lets you either add customizations at the hub side (or) at the node side. Lets take an example use case to understand this customization need a bit more in detail.

For the purpose of debugging, we would like to have access to the logs generated by all the nodes. The only problem is that we would need to enable logging into each of the machines where the nodes run. Instead of this we can build a custom solution by adding servlets at both the hub and the node.

This would be a very good use case for us to customize the Hub. We first start off by creating a custom servlet. All servlets (be it for the Hub or for the Node) can be created by either extending:

1. org.openqa.grid.web.servlet.RegistryBasedServlet (or)
2. javax.servlet.http.HttpServlet.

Extend org.openqa.grid.web.servlet.RegistryBasedServlet when you need access to the internals of the Hub (for e.g., its org.openqa.grid.internal.Registry which is the heart of the Hub) and extendjavax.servlet.http.HttpServlet if you don’t need access to the Hub internals.

So for our example, lets say we are creating two servlets.

1. A servlet ( lets call it as org.openqa.demo.AllNodes ) to be injected at the hub. When this servlet is invoked, it would list out all the nodes that are registered to the Hub and
2. A servlet ( lets call it as org.openqa.demo.NodeLog ) to be injected into the node. When this servlet is invoked, it would read the logs from the node and serve it as a web page. For the sake of simplicity we are not going to be deliving into how to have the node redirect all its logging to a log file.

Now you need to build a jar file (lets assume its called myservlets.jar) that contains both the above mentioned classes (AllNodes and NodeLog).

From the directory where both myservlets.jar and selenium-server-standalone-2.44.0.jar exist, start the hub by running the command

java -cp \*:. org.openqa.grid.selenium.GridLauncher -role hub -servlets org.openqa.demo.AllNodes

This command causes the Grid to be spawned and our new servlet gets added to the Hub. It can be accessed via http://localhost:4444/grid/admin/AllNodes

From the directory where both myservlets.jar and selenium-server-standalone-2.44.0.jar exist, start the node by running the command

java -cp \*:. org.openqa.grid.selenium.GridLauncher -role node -hub http://localhost:4444/grid/register -servlets org.openqa.demo.AllNodes

This command causes the Node to be spawned and our new servlet gets added to the node. It can be accessed via http://xxx:5555/extra/NodeLog where xxx represents the machine name/ip where the node is running.

Points to remember:

1. Assuming that the Hub is running on port 4444 all servlets added to the hub are accessible under http://xxx:4444/admin/ path and
2. Assuming that the node is running on port 5555 all serlvets added to the node are accessible under http://xxx:5555/extra/ path.

**Adding a custom Proxy**

There can be situations wherein you would like to build some custom logic at each of the nodes such as “auto restart” (or) “start/stop video recording” etc., In these kind of situations you would go about building your own custom proxy and then injecting it into the Grid system.

Here’s how you would go about doing it.

1. Start off by extending org.openqa.grid.selenium.proxy.DefaultRemoteProxy and define your custom functionality in it.
2. Build a jar ( lets call the jar as myproxy.jar ) that includes your custom proxy ( For the sake of simiplicity lets assume our custom proxy is going to be called asorg.openqa.grid.MyProxy ).
3. From the directory where both myproxy.jar and selenium-server-standalone-2.44.0.jar exist, start the hub by running the command.

java -cp \*:. org.openqa.grid.selenium.GridLauncher -role hub

1. From the directory where both myproxy.jar and selenium-server-standalone-2.44.0.jar exist, start the node by running the command.

java -cp \*:. org.openqa.grid.selenium.GridLauncher -role node -hub http://localhost:4444/grid/register -proxy org.openqa.grid.MyProxy

Points to remember:

1. When working with custom proxies, make sure that proxy only adds custom functionality on top of what is defined in org.openqa.grid.selenium.proxy.DefaultRemoteProxy.
2. Make sure that the jar that contains your custom proxy class is available in the classpath of both the Grid Hub and the Node ( this was why we resorted to usingjava -cp instead of using java -jar).

Getting Command-Line Help

The Selenium-Server provides listings of available options with a brief description of each. Currently (summer 2012), the command-line help has some oddities, but it can be helpful if you know where to look and how to interpret the information.

The Selenium-Server provides two distinct functions, that of the Selenium-RC server and that of Selenium-Grid. These were likely written by different Selenium teams, and therefore the command-line help for each function has ended up in two different places. And, for the new user, it may not be apparent at first which of these two you are viewing.

If you simply pass a -h option as you might first assume, you get the Selenium-RC Server options but not those for Selenium-Grid.

java -jar selenium-server-standalone-2.44.0.jar -h

This would give you Selenium-RC’s server options. If you want the command-line help for Selenium-Grid, you first use the -hub or -node options to tell Selenium-Server you’re interested in Selenium-Grid, and then follow with a -h.

java -jar selenium-server-standalone-2.44.0.jar -role node -h

Or, for that matter, just pass a garbage argument to the -role node as follows.

java -jar selenium-server-standalone-2.44.0.jar -role node xx

You will first see “INFO...” and an “ERROR” but below that you’ll get the command-line options for Selenium-Grid. We won’t list the whole output here since it’s rather long, but the first few lines look like this.

Jul 19, 2012 10:10:39 AM org.openqa.grid.selenium.GridLauncher main

INFO: Launching a selenium grid node

org.openqa.grid.common.exception.GridConfigurationException: You need to specify a hub to register to using -hubHost X -hubPort 5555. The specified config was -hubHost null -hubPort 4444

at org.openqa.grid.common.RegistrationRequest.validate(RegistrationRequest.java:610)

at org.openqa.grid.internal.utils.SelfRegisteringRemote.startRemoteServer(SelfRegisteringRemote.java:88)

at org.openqa.grid.selenium.GridLauncher.main(GridLauncher.java:72)

Error building the config :You need to specify a hub to register to using -hubHost X -hubPort 5555. The specified config was -hubHost null -hubPort 4444

Usage :

-hubConfig:

(hub) a JSON file following grid2 format.

-nodeTimeout:

(node) <XXXX> the timeout in seconds before the hub

automatically ends a test that hasn't had aby activity than XX

sec.The browser will be released for another test to use.This

typically takes care of the client crashes.

Common Errors

**Unable to acess the jarfile**

Unable to access jarfile selenium-server-standalone-2.44.0.jar

This error can occur when starting up either a hub or node. This means Java cannot find the selenium-server jar file. Either run the command from the directory where the selenium-server-XXXX.jar file is stored, or specify an explicit path to the jar.