Working With Selenium Grid

[Selenium Grid Tutorial: Step by Step Guide with Example 1](#_Toc491947592)

[What is Selenium Grid? 2](#_Toc491947593)

[When to Use Selenium Grid? 2](#_Toc491947594)

[Grid 1.0 Vs Grid 2.0 2](#_Toc491947595)

[Selenium Grid Architecture 3](#_Toc491947596)

[How to Set Up Selenium Grid? 3](#_Toc491947597)

[Designing Test Scripts That Can Run on the Grid 7](#_Toc491947598)

[Using the DesiredCapabilites Object 7](#_Toc491947599)

[Using the RemoteWebDriver Object 7](#_Toc491947600)

[Running a Sample Test Case on the Grid 8](#_Toc491947601)

[Summary 9](#_Toc491947602)

[How to Setup a Hub and Node for Selenium Grid 9](#_Toc491947603)

[Execute webdriver tests in Parallel using selenium Grid 14](#_Toc491947604)

[Configure Selenium Grid using Docker 21](#_Toc491947605)

[How Docker helps? 21](#_Toc491947606)

[What is Docker ? 21](#_Toc491947607)

[Configure Selenium Grid in Docker Containers 22](#_Toc491947608)

[View test execution using debug nodes 25](#_Toc491947609)

[How to increase nodes ? 25](#_Toc491947610)

[What is Docker Compose ? 26](#_Toc491947611)

[Increase Nodes using Docker-compose 28](#_Toc491947612)

[Selenium Grid Hub and Node configuration using JSON 29](#_Toc491947613)

[Configuring hub using below JSON:- 29](#_Toc491947614)

[Start hub using below command :- 30](#_Toc491947615)

[Configuring nodes using below JSON :- 31](#_Toc491947616)

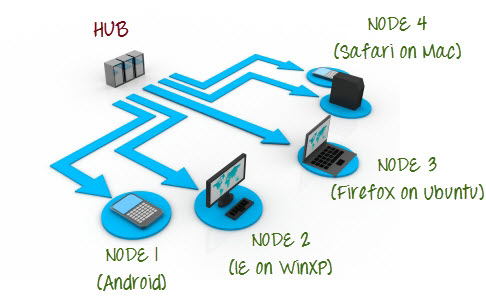
[Start node using below command :- 32](#_Toc491947617)

# Selenium Grid Tutorial: Step by Step Guide with Example

## What is Selenium Grid?

Selenium Grid is a part of the Selenium Suite that specializes in running multiple tests across different browsers, operating systems, and machines in parallel.

Selenium Grid has 2 versions - the older Grid 1 and the newer Grid 2. We will only focus on Grid 2 because Grid 1 is gradually being deprecated by the Selenium Team.

[](https://cdn.guru99.com/images/hub_and_nodes.jpg)

Selenium Grid uses a hub-node concept where you only run the test on a single machine called a **hub**, but the execution will be done by different machines called **nodes**.

## When to Use Selenium Grid?

You should use Selenium Grid when you want to do either one or both of following:

* **Run your tests against different browsers, operating systems, and machines all at the same time.**This will ensure that the application you are[Testing](https://www.guru99.com/software-testing.html)is fully compatible with a wide range of browser-O.S combinations.
* **Save time in the execution of your test suites**. If you set up Selenium Grid to run, say, 4 tests at a time, then you would be able to finish the whole suite around 4 times faster.

## Grid 1.0 Vs Grid 2.0

Following are the main differences between Selenium Grid 1 and 2.

|  |  |
| --- | --- |
| **Grid 1** | **Grid 2** |
| Selenium Grid 1 has its own remote control that is different from the Selenium RC server. They are two different programs. | Selenium Grid 2 is now bundled with the Selenium Server jar file |
| You need to install and configure[Apache](https://www.guru99.com/apache.html)Ant first before you can use Grid 1. | You do not need to install[Apache](https://www.guru99.com/apache.html)Ant in Grid 2. |
| Can only support Selenium RC commands/scripts. | Can support both Selenium RC and WebDriver scripts. |
| You can only automate one browser per remote control. | One remote control can automate up to 5 browsers. |

## Selenium Grid Architecture

The Hub

* The hub is the central point where you load your tests into.
* There should only be one hub in a grid.
* The hub is launched only on a single machine, say, a computer whose O.S is Windows 7 and whose browser is IE.
* The machine containing the hub is where the tests will be run, but you will see the browser being automated on the node.

The Nodes

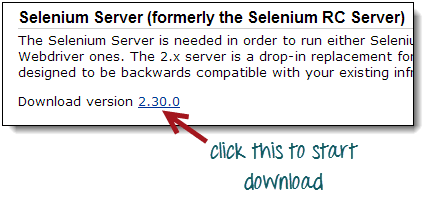
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* There can be one or more nodes in a grid.
* Nodes can be launched on multiple machines with different platforms and browsers.
* The machines running the nodes need not be the same platform as that of the hub.

## How to Set Up Selenium Grid?

In this section, you will use 2 machines. The first machine will be the system that will run the hub while the other machine will run a node. For simplicity, let us call the machine where the hub runs as "Machine A" while the machine where the node runs will be "Machine B." It is also important to note their IP addresses. Let us say that Machine A has an IP address of 192.168.1.3 while Machine B has an IP of 192.168.1.4.

**Step 1**

Download the Selenium Server by [here](http://docs.seleniumhq.org/download/).

<https://selenium-release.storage.googleapis.com/index.html?path=2.53/>[](https://cdn.guru99.com/images/download_selenium_server.png)

**Step 2**

You can place the Selenium Server .jar file anywhere in your HardDrive. But for the purpose of this tutorial, place it on the C drive of both Machine A and Machine B. After doing this, you are now done installing Selenium Grid. The following steps will launch the hub and the node.

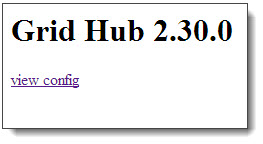
**Step 3**

* We are now going to launch a hub. Go to Machine A. Using the command prompt, navigate to the root of Machine A's - C drive, because that is the directory where we placed the Selenium Server.
* On the command prompt, type **java -jar selenium-server-standalone-2.30.0.jar -role hub**
* The hub should successfully be launched. Your command prompt should look similar to the image below

[](https://cdn.guru99.com/images/launch_hub.jpg)

**Step 4**

Another way to verify whether the hub is running is by using a browser. Selenium Grid, by default, uses Machine A's port 4444 for its web interface. Simply open up a browser and go to <http://localhost:4444/grid/console>

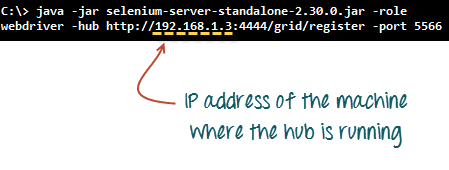
[](https://cdn.guru99.com/images/view_hub_from_browser.jpg)

Also, you can check if Machine B can access the hub's web interface by launching a browser there and going to where "iporhostnameofmachineA" should be the IP address or the hostname of the machine where the hub is running. Since Machine A's IP address is 192.168.1.3, then on the browser on Machine B you should type [http://192.168.1.3:4444/grid/console](http://192.168.1.11:4444/grid/console)

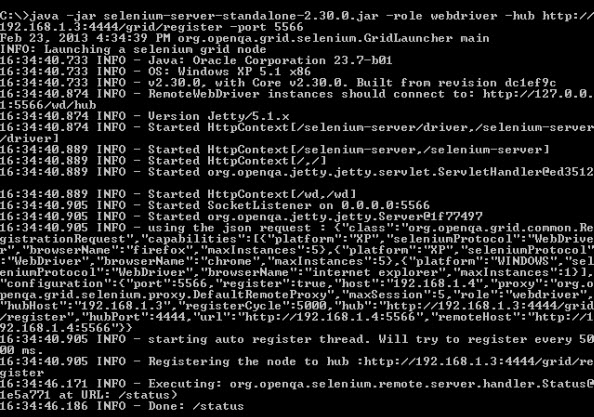
**Step 5**

* Now that the hub is already set up, we are going to launch a node. Go to Machine B and launch a command prompt there.
* Navigate to the root of Drive C and type the code below. We used the IP address 192.168.1.3 because that is where the hub is running. We also used port 5566 though you may choose any free port number you desire.
* NOTE: You now have to give path to the Gecko driver if using Firefox. Here is updated code that needs to be used

java -D webdriver.gecko.driver="C:\geckodriver.exe" -jar selenium-server-standalone-3.4.0.jar -role webdriver -hub http://192.168.1.3:4444/grid/register -port 5566

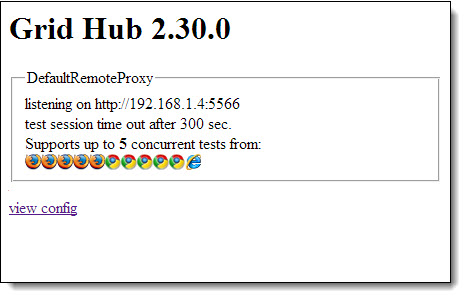
[](https://cdn.guru99.com/images/grid_2_launch_node.png)

* When you press Enter, your command prompt should be similar to the image below.

[](https://cdn.guru99.com/images/after_node_is_launched.jpg)

**Step 6**

Go to the Selenium Grid web interface and refresh the page. You should see something like this.

[](https://cdn.guru99.com/images/view_node_from_browser.jpg)

At this point, you have already configured a simple grid. You are now ready to run a test remotely on Machine B.

## Designing Test Scripts That Can Run on the Grid

To design test scripts that will run on the grid, we need to use **DesiredCapabilites** and the **RemoteWebDriver** objects.

* **DesiredCapabilites**is used to set the type of **browser** and **OS**that we will automate
* **RemoteWebDriver**is used to set which node (or machine) that our test will run against.

To use the **DesiredCapabilites**object, you must first import this package

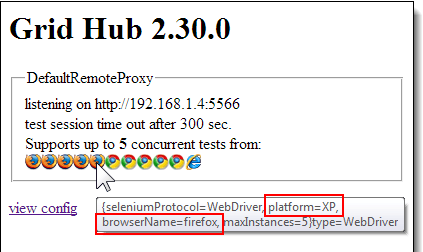
[Selenium Grid Tutorial: Step by Step Guide with Example](https://cdn.guru99.com/images/import_desired_capabilites.png)

To use the **RemoteWebDriver**object, you must import these packages.

[Selenium Grid Tutorial: Step by Step Guide with Example](https://cdn.guru99.com/images/import_RemoteWebDriver.png)

## Using the DesiredCapabilites Object

Go to the Grid's web interface and hover on an image of the browser that you want to automate. Take note of the **platform,** and the **browserName**showed by the tooltip.

[](https://cdn.guru99.com/images/ToolTip(1).png)

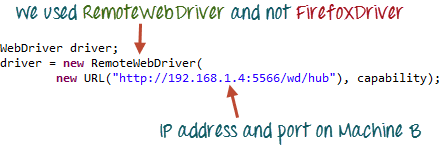
In this case, the platform is "XP" and the browserName is "Firefox."

We will use the platform and the browserName in our WebDriver as shown below (of course you need to import the necessary packages first).

[Selenium Grid Tutorial: Step by Step Guide with Example](https://cdn.guru99.com/images/DesiredCapabilities_code.png)

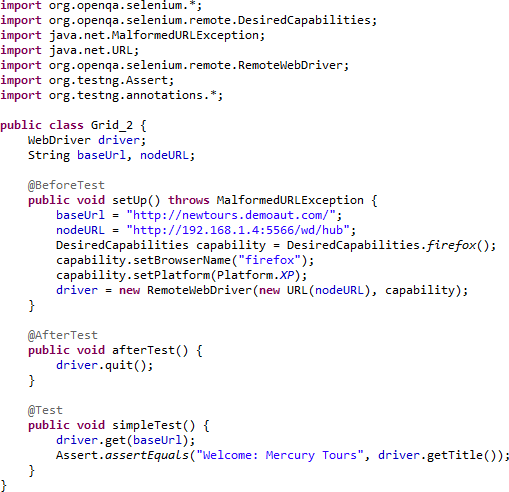
## Using the RemoteWebDriver Object

Import the necessary packages for RemoteWebDriver and then pass the DesiredCapabilities object that we created above as a parameter for the RemoteWebDriver object.

[](https://cdn.guru99.com/images/RemoteWebDriver_code.png)

## Running a Sample Test Case on the Grid

Below is a simple WebDriver TestNG code that you can create in Eclipse on Machine A. Once you run it, automation will be performed on Machine B.

[](https://cdn.guru99.com/images/whole_code.png)

 The test should pass.

|  |  |
| --- | --- |
| [Selenium Grid Tutorial: Step by Step Guide with Example](https://cdn.guru99.com/images/result_1.png) | [Selenium Grid Tutorial: Step by Step Guide with Example](https://cdn.guru99.com/images/result_2.png) |

## Summary

* Selenium Grid is used to run multiple tests simultaneously on different browsers and platforms.
* Grid uses the hub-node concept.
* The hub is the central point wherein you load your tests.
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* To install Selenium Grid, you only need to download the Selenium Server jar file - the same file used in running Selenium RC tests.
* There are 2 ways to verify if the hub is running: one was through the command prompt, and the other was through a browser
* To run test scripts on the Grid, you should use the DesiredCapabilities and the RemoteWebDriver objects.
* DesiredCapabilites is used to set the type of browser and OS that we will automate
* RemoteWebDriver is used to set which node (or machine) that our test will run against.

# How to Setup a Hub and Node for Selenium Grid

Selenium Grid allows us to execute our tests in multiple machines (physical / virtual) and multiple browsers with different versions, which dramatically speeds up test execution and helps in reducing total amount of time required for test execution.

**For example**, if we have a script that takes 100 minutes to execute sequentially , we could break that down to 10 short tests script run across 10 machines, and can complete them in 10 minutes without copying your test code to the other machine.

To get started with [Selenium Grid](https://code.google.com/p/selenium/wiki/Grid2), make sure you have Java installed and configured it and For selenium, you need to download [selenium server](http://www.seleniumhq.org/download/) and place it in a directory.

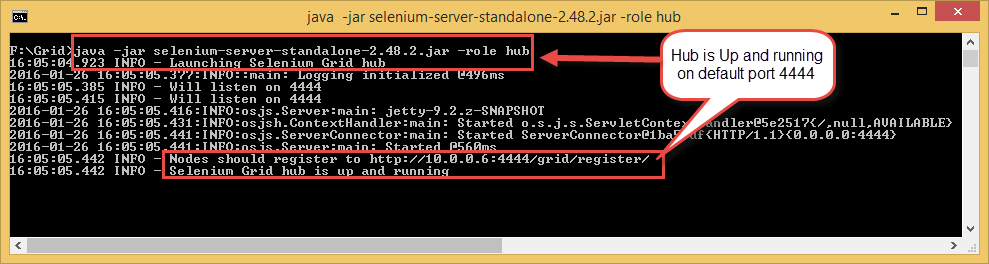
Selenium Grid hub/node can be configured in 2 different ways, one is by specifying command line parameters, and the other way is by specifying a [JSON config file](http://www.seleniumeasy.com/selenium-tutorials/configure-selenium-grid-using-json-config-file).

A grid consists of a single hub, and one or more nodes, Hub and Node are the two main elements that you come across when using grid

**Hub** the Hub is the central point which will receive all the test requests along with information on which browser, platform (i.e. WINDOWS, LINUX, etc) and where the test should be run. Based on the request received, it will distribute them to the registered nodes.

To start a hub with default parameters, we can run the below command from a command-line. Just navigate to the directory where your selenium jar file is available and execute the below statement. You can open command prompt from the same folder using 'Press Shift and Right Click' you see an option 'Open command window here'.

java -jar selenium-server-standalone-2.48.2.jar -role hub



In the above statement, we have started Hub using default parameters, So the default port will be 4444 and Hub listen for new requests is port 4444. This is why port 4444 was used in the URL for locating the hub. You can also change the default port, by adding the optional parameter -port when you run the command example: -port 5555.

After starting the hub, we can view the status of the hub by opening any browser window and navigating to: [*http://localhost:4444/grid/console*](http://localhost:4444/grid/console). If you have used any other port, you need to mention that port value instead of default port 4444.

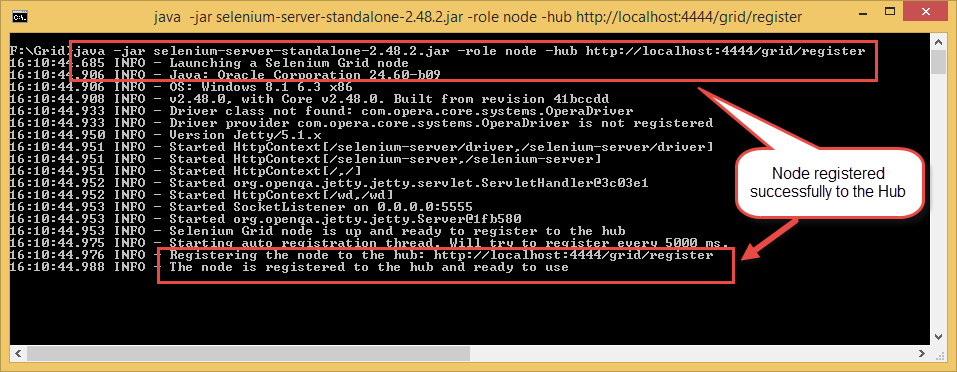


**Nodes** are where our tests will run, each Node is machine (can be a physical machine / virtual machine) that we register with the Hub, when we register Node, Hub will get to know about the node, and it will display browser and configuration details of the node that we used to register node with parameters.

Below is the command ***to register node with a Hub***. If we are not specifying any parameters when starting node, it defaults to 5555 whenever **"-role"**option is provided and is not hub.

java -jar selenium-server-standalone-2.48.2.jar -role node -hub http:*//localhost:4444/grid/register*

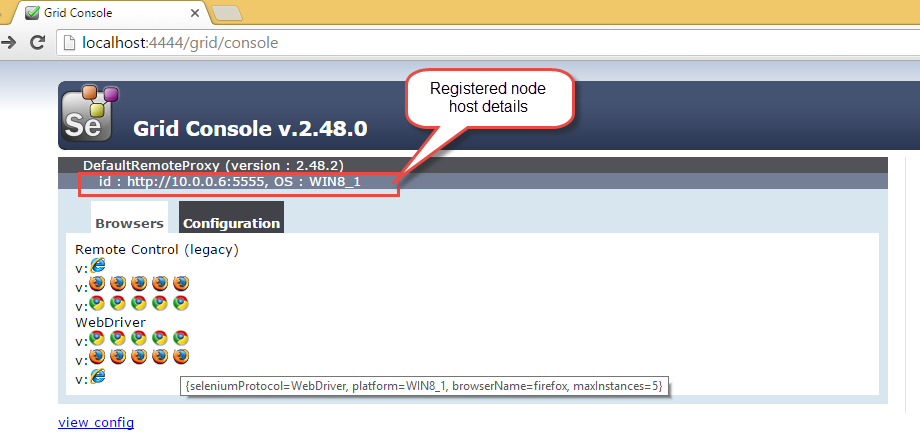
After executing above command, you should see something like below. When the hub is running in the same machine, we use ‘localhost’ for node. If Hub and Node are running on separate machines, we have to register Node using the hostname of the remote machine running the hub.



By default, when we start the node, it starts total 11 browsers : 5 Firefox, 5 Chrome and 1 Internet Explorer and it has same set of browsers for Selenium Remote Control (legacy). The maximum number of concurrent tests is default to 5.

We can change this and other browser settings as well by passing the parameters to each -browser switch (each switch represents a node based on your parameters). If you use the -browser parameter, the default browsers will be ignored and only what you specify command line will be used. The node can be configured in two different ways, One is by specifying command line parameters, the other is by specifying by a json file. We will discuss more in detail on these parameters in next articles.

For now, this is how it looks when we register without specifying any parameters



When you mouse hover on the browser icons, it will show config information, here "seleniumProtocol": "Selenium" which provides the mechanism for Selenium RC (Remote control / Selenium 1) and "seleniumProtocol": "WebDriver" is for Selenium WebDriver. After registering the node, You can see the difference in grid console as above.

We have done with starting the Hub and Registering Nodes with the Hub. Now we need to run our tests with Selenium Grid, For webdriver nodes, we need to use the [RemoteWebDriver](https://code.google.com/p/selenium/wiki/RemoteWebDriver" \t "_blank) and [DesiredCapabilities](https://code.google.com/p/selenium/wiki/DesiredCapabilities" \t "_blank) object to define which browser, version of the browser and platform (OS - Windows / LINUX etc) that we want to run our tests.

Based on preferences that we set in the DesiredCapabilities instance, the Hub will point our tests to a node that matches with these preferences. If we specify capabilities that do not exist on our grid then there will be no match and the test will fail to run.

**Why we need to use RemoteWebdriver Not the webdriver ?**

If we use driver (FirefoxDriver / ChromeDriver / or other) not RemoteWebDriver, it will just assume that the communication to the browser is local. *Example: - Webdriver driver = new FirefoxDriver();* Using this, driver will access Firefox browser which is available on the local machine.

If we use RemoteWebDriver, it requires us to specify where the Selenium Server is located and on which web browser we want to execute our tests. For example,

WebDriver driver = **new** RemoteWebDriver(**new** URL("[http://localhost:4444/wd/hub"](http://localhost:4444/wd/hub)), DesiredCapabilities.firefox());

Here in above statement , it is specified that Selenium Server is running on localhost with the default port 4444 and execute on firefox browser. In the same fashion, we can run selenium server on one machine as Hub and execute selenium tests on other machine by registering to the node by specifying parameters.

In the below example, Hub will point the test to a node which is running on the Windows machine (local machine) with Firefox browser as my node is executing in windows machine and capabilities specified is Only firefox browser.

**package** com.test;

**import** java.net.MalformedURLException;

**import** java.net.URL;

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

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

**import** org.testng.Assert;

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

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

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

**public** **class** **GridExampleTest** {

**public** RemoteWebDriver driver;

**public** **static** String appURL = "[http://www.google.com";](http://www.google.com/)

@BeforeClass

**public** **void** **setUp**() **throws** MalformedURLException {

DesiredCapabilities capabilities = DesiredCapabilities.firefox();

driver = **new** RemoteWebDriver(**new** URL("[http://localhost:4444/wd/hub"](http://localhost:4444/wd/hub)), capabilities);

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

}

@Test

**public** **void** **testGooglePageTitleInIEBrowser**() {

System.out.println("\*\*\* Navigation to Application \*\*\*");

driver.navigate().to(appURL);

String strPageTitle = driver.getTitle();

System.out.println("\*\*\* Verifying page title \*\*\*");

Assert.assertTrue(strPageTitle.equalsIgnoreCase("Google"), "Page title doesn't match");

}

@AfterClass

**public** **void** **closeBrowser**() {

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

driver.quit();

}

}

}

In the above test, BeforeClass method open firefox browser and , verifies page title of google. Once done, it will quit the remote webdriver.

Below is the testng.xml file :

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

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

<**suite** name="Main Test Suite" verbose="1">

    <**test** name="TestNG Grid">

        <**classes**>

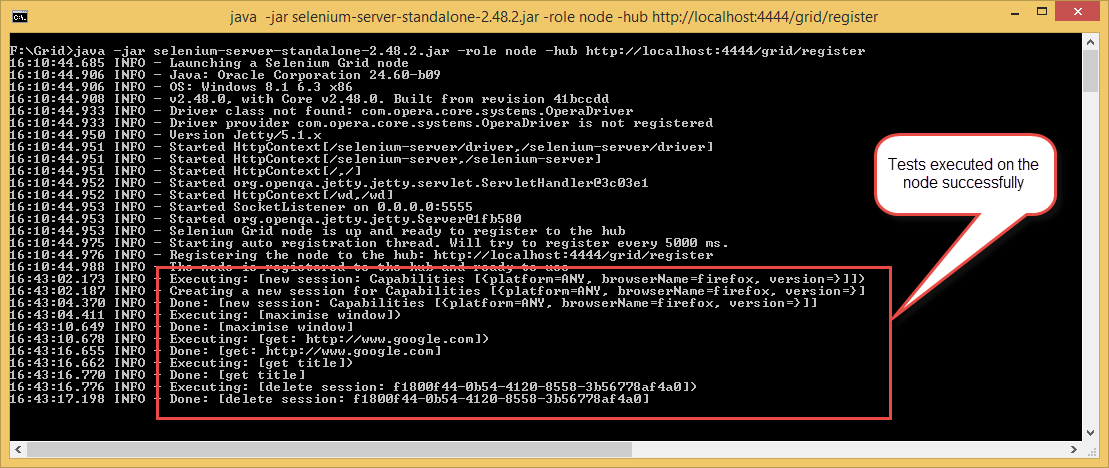
        <**class** name="com.test.GridExampleTest"/>

        </**classes**>

    </**test**>

</**suite**>

After executing test as TestNG, you can view the console for details such as on which browser the tests are executed along with version and platform details if specified any.



Hope you have successfully configured Selenium Grid on your machine. We will discuss configuring grid with multiple machines with examples later.

# Execute webdriver tests in Parallel using selenium Grid

In previous article we have seen [configuring selenium grid](http://www.seleniumeasy.com/selenium-tutorials/how-to-configure-selenium-grid) and execute a simple test on firefox browser. In this tutorial we will see 'Parallel execution of tests' using selenium grid and execute tests on firefox and chrome browser.

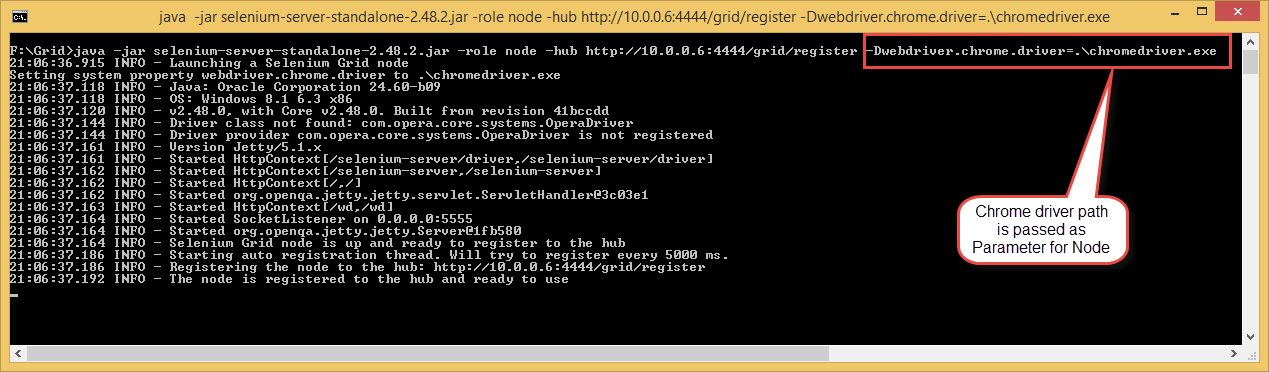
We will register multiple nodes to the Hub and execute tests in parallel. In the below example we will register a node in the same local machine where hub is running and other node in remote machine.

We will register ***node 1*** in local machine with the below command:  
 **java -jar selenium-server-standalone-2.48.2.jar -role node -hub**[**http://localhost:4444/grid/register**](http://localhost:4444/grid/register)

We will register ***node 2*** with the below command in remote machine. In the below command, need to pass a parameter by passing JVM properties using the -D flag along with chrome driver path , so that when ever there is a request to execute in chrome driver, Hub will send the request to this node. And here we have to mention the IP address of the machine where the hub is running as we are starting this node in remote machine.

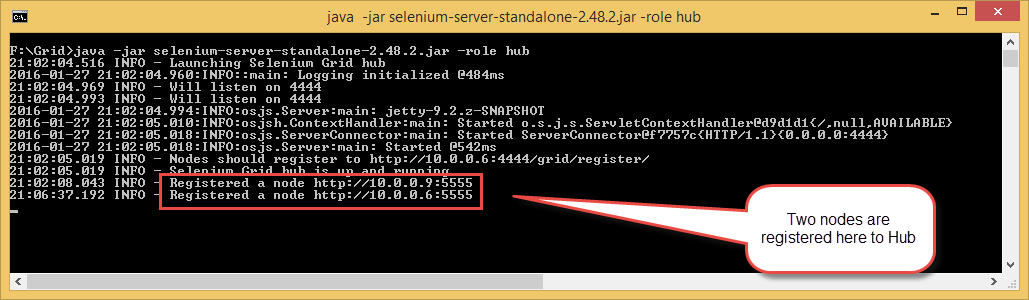
**java -jar selenium-server-standalone-2.48.2.jar -role node -hub**[**http://10.0.0.6:4444/grid/register**](http://10.0.0.6:4444/grid/register)**-Dwebdriver.chrome.driver=.\chromedriver.exe**

After executing the above command, console should look like below:



When a Hub receives request to execute test in Chrome browser and If we don't specify chrome driver path for node, it will throw an exception as "Exception: The path to the chromedriver executable must be set by the webdriver.chrome.driver system property; for more information, see <http://code.google.com/p/selenium/wiki/ChromeDriver>. The latest version can be downloaded from <http://code.google.com/p/chromedriver/downloads/list>".

Now when these two nodes are registered to the hub, hub console should look like below :



And the Grid console should look like below which shows configuration details of the nodes with IP address. In the below image below image Node 1 is running in Windows \* machine (WIN8\_1) and Node 2 is running in Windows7 machine.



#### Let us now take the example and execute tests in parallel.

In order to do this, we will create two classes which has multiple @Test methods in it. And a Browser class which invokes the remote webdriver based on the browser parameter that we pass in testng.xml file

**First create a class as 'Browser.java'**

**package** com.test;

**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) **throws** MalformedURLException {

**return** **new** RemoteWebDriver(**new** URL("[http://10.0.0.6:4444/wd/hub"](http://10.0.0.6:4444/wd/hub)), getBrowserCapabilities(browser));

}

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

**switch** (browserType) {

**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();

}

}

}

We will call 'getDriver' method which will intern call 'getBrowserCapabilities' based on browser parameter from the below two classes. If we pass 'chrome' as parameter, it is invoke chromedriver

#### Let us create a class as 'ParallelTestA.java' as below:

**package** com.test;

**import** java.net.MalformedURLException;

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

**import** org.testng.Assert;

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

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

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

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

**public** **class** **ParallelTestA** {

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

**public** **static** String appURL = "[http://www.google.com";](http://www.google.com/)

@BeforeClass

@Parameters({ "browser" })

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

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

driver = Browser.getDriver(browser);

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

}

@Test

**public** **void** **testGooglePageTitleInFirefox**() {

driver.navigate().to(appURL);

String strPageTitle = driver.getTitle();

Assert.assertTrue(strPageTitle.equalsIgnoreCase("Google"), "Page title doesn't match");

}

@AfterClass

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

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

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

driver.quit();

}

}

}

#### Let us create a class as 'ParallelTestB.java' as below:

**package** com.test;

**import** java.net.MalformedURLException;

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

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

**import** org.testng.Assert;

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

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

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

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

**public** **class** **ParallelTestB** {

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

**public** **static** String appURL = "[http://www.google.com";](http://www.google.com/)

@BeforeClass

@Parameters({ "browser" })

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

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

driver = Browser.getDriver(browser);

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

}

@Test

**public** **void** **testGooglePageTitleInChrome**() {

driver.navigate().to("[http://www.google.com"](http://www.google.com/));

String strPageTitle = driver.getTitle();

Assert.assertTrue(strPageTitle.equalsIgnoreCase("Google"), "Page title doesn't match");

}

@Test

**public** **void** **testSearchGoogle**() {

System.out.println("Opening Google..");

driver.navigate().to(appURL);

driver.findElement(By.name("q")).sendKeys("Selenium Easy Grid Tutorials");

driver.findElement(By.name("btnG")).click();

}

@AfterClass

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

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

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

driver.quit();

}

}

}

**Now to execute these tests, we need to create testng.xml file as below and set parallel="tests" with parameter browser for each test**

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

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

<**suite** name="Main Test Suite" parallel="tests" verbose="1">

    <**test** name="Grid firefox Test">

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

        <**classes**>

        <**class** name="com.test.ParallelTestA"/>

        </**classes**>

    </**test**>

    <**test** name="Grid chrome Test">

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

        <**classes**>

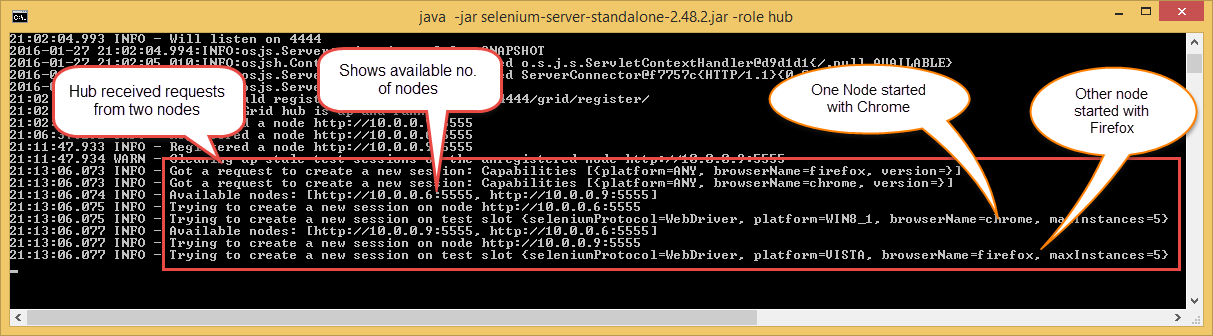
        <**class** name="com.test.ParallelTestB"/>

        </**classes**>

    </**test**>

</**suite**>

Once we execute the above code, hub console will display information such as on 'number of nodes available', on which node it has started executing the tests etc.. as below :

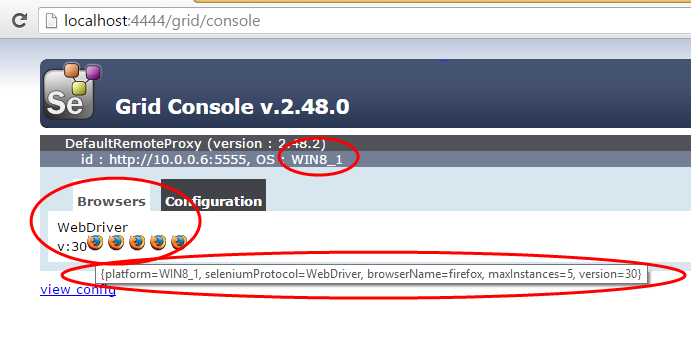


There are many other parameters for browser settings that we can pass when registering node to hub. When ever we use **-browser** parameter, the default browsers will be ignored and only what we specify in command line will be used.

Example :

-browser browserName=firefox,version=30,maxInstances=5,platform=WIN8\_1

After executing the above command, if you check the grid console, it will just show the browserName, browser version, Max instances (number of instances of same version of browser you can run over the Remote System) and the platform (OS running on the node) that we have used to register the node. Grid console looks like below :



If the remote machine has multiple versions of Firefox, We can map the location of each binary to a particular version on the same machine to execute with multiple versions as below:

-browser browserName=firefox,version=40,firefox\_binary=c:\Program Files\firefox30\firefox,maxInstances=3,platform=WINDOWS -browser browserName=firefox,version=41,firefox\_binary=c:\Program Files\firefox40\firefox,maxInstances=3,platform=WINDOWS

# Configure Selenium Grid using Docker

Generally, to run our [Selenium tests in parallel](http://www.seleniumeasy.com/selenium-tutorials/parallel-execution-in-selenium-grid), we use selenium grid and distribute tests across multiple physical or virtual machines which will reduce the time required for running tests.

To run tests in parallel, we need to [configure Selenium Grid](http://www.seleniumeasy.com/selenium-tutorials/how-to-configure-selenium-grid) with Hub and Nodes where hub is the central point which will receive test requests along with configurations or capabilities. Based on the request received, hub will distribute tests to the registered nodes.

Most of the organisations execute their Selenium scripts on virtual machines. Based on number of scripts, they need to increase the node machines. When the nodes increases maintaining the Grid environment will become a problem in setting up required browsers / OS combinations across these virtual machines.

And also, there are times were issues like **Browser Crashes**, **nodes running out of memory**, **Java processes run out of memory** etc kills our time in restating node machines/hub.

There are few companies who uses Cloud-based platform for automated testing of web/mobile applications like [SauceLabs](https://saucelabs.com/" \t "_blank) / [Browserstack](https://www.browserstack.com/" \t "_blank) / [CrossBrowserTesting](https://crossbrowsertesting.com/" \t "_blank) but yes these are expensive.

## How Docker helps?

Docker is a lightweight container which can make you to run distributed applications in a minute. Docker Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

Where as each Virtual Machine includes a full copy of an operating system, one or more applications with necessary binaries and libraries which consumes more space and makes slow to boot.

## What is Docker ?

Docker is an open platform for developers and system administrators to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

In other words, Docker is a simple container that let's you to specify a complete package of components needed to run a software.

We will not discuss much on Docker as there are many articles available on Docker and its architecture.

We assume that you have already installed docker and is up running, if not please [install Docker](https://docs.docker.com/engine/installation/). Docker can be installed on Windows, macOS, Linux, and Cloud platforms. Please check the requirements and download based on environment.

Once the installation is completed, please test your installation by running the below command:

**docker** info

The above command displays system wide information regarding the Docker installation like number of containers and images etc.

## Configure Selenium Grid in Docker Containers

To get Selenium Grid up and running, we need to do below steps in Hub/Nodes machines

**Install Java**  
**Install Required Browsers (versions if needed)**  
**Selenium server jars**

Docker has many built in ways to quickly create a grid, scale nodes up or down when ever required with a single command.

Selenium has also made a set of Docker images which are available on Docker Hub. We have Selenium Grid, and the browser images - Chrome and Firefox (for now), also with the debug versions which will allow us to access and view test execution using VNC.

*A Docker image is a read-only template with instructions for creating a Docker container.*

A Docker Hub contains images from official images from organizations like Selenium, Google, Microsoft and a lot more. You can find required images by browsing the Docker Hub. Refer [Selenium Docker Hub](https://hub.docker.com/r/selenium/) and [Selenium Docker Github Page](https://github.com/SeleniumHQ/docker-selenium)

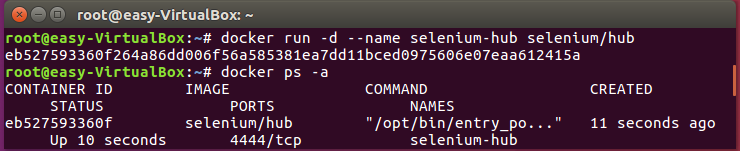
In order to configure the hub on docker container, we need to pull Selenium Hub from Docker repository using below command:-

docker pull selenium/hub

Once the pull is complete, it will show you status as 'Downloaded' at the end.

Now we need to Run the Hub using below command :-

docker run -d --name selenium-hub selenium/hub

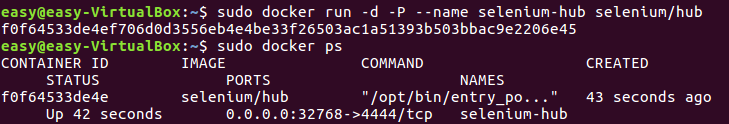


If you want to assign the port, run the below command

docker run -d -p 4545:4444 --name selenium-hub selenium/hub

We can explicitly specify the name for container, else Docker will automatically generate a unique name for each container. In the above command, we have given a name for selenium hub using --name. We can use this name to link node containers to the hub container.

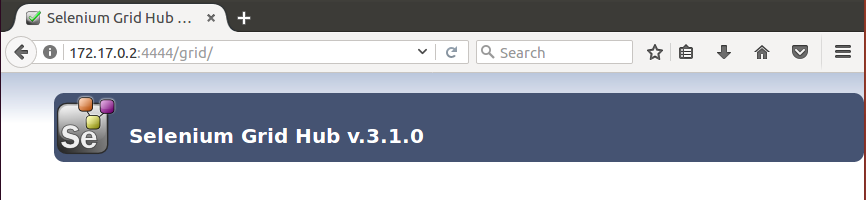
To check if the container has started, run docker ps command and see the status of it.



To check the configuration information, run docker logs

If you see the status as 'Selenium Grid hub is up and running' then its time to start registering the nodes to hub.

We can also now enter the URL in the browser and the see the Grid running up



To register nodes to the Hub, we need to pull nodes from Docker Hub using below commands :-

To get Chrome node docker pull selenium/node-chrome

To get Firefox node docker pull selenium/node-firefox

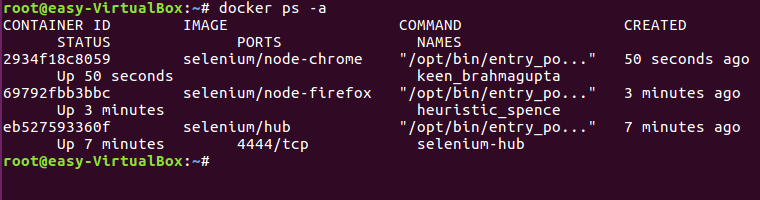
Once the pull complete, we need to link these nodes to the hub container by running below commands :-

Command to link both Chrome and Firefox nodes to selenium-hub

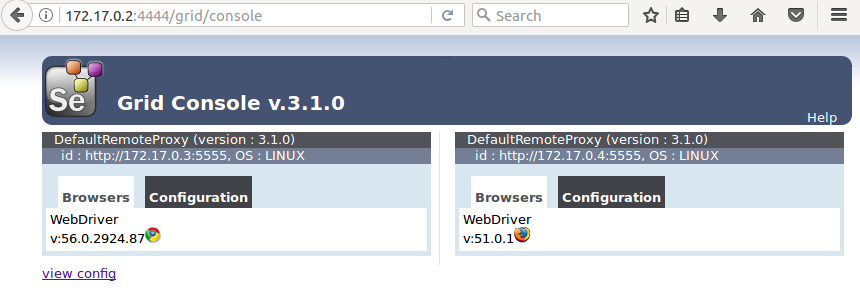
docker run -d --link selenium-hub:hub selenium/node-firefox

docker run -d --link selenium-hub:hub selenium/node-chrome

Now again type docker ps -a command in your terminal, to see a list of containers that are currently active. This will bring up total three containers Selenium Hub, Firefox node, and Chrome node.



We can also see the nodes registered in Grid Console. Enter Grid IP and observe that Chrome and Firefox nodes are connected to Hub.



Done !!! Now your grid configuration is ready. You can execute your selenium automation scripts as you do.

If you want view test execution, we need to pull 'debug' images, which allow us to see the browser executing tests.

**docker** pull selenium/node-chrome-debug

docker pull selenium/node-firefox-debug

By using docker ps –a command, you can see all the Docker containers and their running ports. Now link these nodes to container hub.

docker run –d –**link** selenium-hub:hub selenium/node-chrome-debug

docker run –d –**link** selenium-hub:hub selenium/node-firefox-debug

## View test execution using debug nodes

Now we can view browsers by using VNC viewer. Enter your hub URL and port number of each browser (debug browser chrome/firefox) and Click connect button.

VNC viewer will ask you for a password to show you the browser. Type the defauolt password as 'secret' and then click 'Ok' button.

## How to increase nodes ?

Say, now if you want more nodes (Total 6 nodes - Say 3 Chrome and 3 Firefox), you can create them in the same way as we have done above:

To increase 2 more chrome nodes, run below commands :-

docker run -d --link selenium-hub:hub selenium/node-chrome

docker run -d --link selenium-hub:hub selenium/node-chrome

To increase Firefox nodes, run the below command :-

docker run -d --link selenium-hub:hub selenium/node-firefox

docker run -d --link selenium-hub:hub selenium/node-firefox

Now check the Grid console, it will have total 3 chrome and 3 Firefox instances running.

We have executed multiple commands to get Selenium Grid Hub and nodes up and running. Now Let us have all together and start Selenium Grid By running a single docker-compose command.

## What is Docker Compose ?

*Docker Compose is a tool for defining and running multi-container Docker applications.*

We will create a docker-compose.yml file and see how the images interact with each other. Docker compose uses ".yml" file to create and start all the services defined.

To install Docker Compose, you’ll need to install Docker first. We assume that you have already installed docker, now please [install docker compose](https://docs.docker.com/compose/install/) and proceed.

Below is the command to install docker-compose, check for the latest release and replace version in the command below if necessary.

**sudo** curl -o /usr/local/bin/docker-compose -L "<https://github.com/docker/compose/releases/download/1.11.2/docker-compose->**$(uname -s)**-**$(uname -m)**"

Now set the permissions by running below command :-

sudo **chmod** +**x** /usr/**local**/bin/docker-compose

Then we'll verify that the installation was successful by checking the docker-compose version:

docker-compose *--version*

If the permissions are not set, you may get error as bash: /usr/local/bin/docker-compose: Permission denied

That's all. We are all set to go, to pull and run Selenium Grid Hub and Nodes with a Single command. Docker Compose will download the images, configure, and start them running.

docker-compose up

Before running Docker Compose, we will stop running containers (if there are any) and remove by running below commands.

To Stop Container docker stop containerId

To Remove Container docker rm containerId

To stop all running containers docker stop $(docker ps -q)

To remove all containers docker rm $(docker ps -aq)

We will create a simple Docker-compose file like below and add this to the repository. Below is the simple Docker-compose.yml file to pull, link and run Selenium Grid Hub and nodes.

**hub:**

  image: selenium/hub

  ports:

    - "4444:4444"

**firefox:**

  image: selenium/node-firefox

  links:

    - hub

**chrome:**

  image: selenium/node-chrome

  links:

    - hub

In the above docker compose file, we have defined selenium hub entity, pointed image name and assigned the port. We have defined and linked two nodes firefox-node and chrome-node. Just navigate to the location where docker-compose.yml file is located and run the below command.

docker-compose up -d

After running above command, check 'docker ps' or open Grid console to see grid Hub and nodes running.

When there are huge tests that need to run parallel, two nodes may be not enough. Docker-compose has a great feature which allows us to scale number of similar containers with single command.

## Increase Nodes using Docker-compose

If you want to increase the number of chrome nodes to three, enter command:

docker-compose scale chrome=3

Again if you want to increase the number of firefox nodes to 3, enter below command:

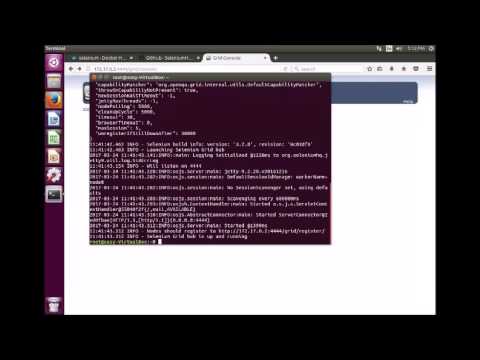
docker-compose scale firefox=3

To check if all the 6 nodes registered to the Grid Hub, run the below command with selenium-hub container id :-

docker logs <**conatinerId**>

This will show you all the containers that are registered to the Hub. It just took few minutes/commands to have Selenium Grid up and Running successfully.

Part 1: Step by step video to configure selenium grid using Docker



Part 2: Step by step video to Setup Selenium Grid using Docker-Compose



Hope this helps you to setup Grid on your containers and run selenium tests with simple command.

# Selenium Grid Hub and Node configuration using JSON

Selenium Grid is used to speed up the execution by using multiple machines (multiple browsers with different versions, and browsers running on different operating systems) and run tests in parallel. And also to reduce the time spent for running the test suite after developers check-in their code.

If you just want to get started using selenium grid, you can use simple commands as we discussed in [previous article](http://www.seleniumeasy.com/selenium-tutorials/how-to-configure-selenium-grid). But if you need more control with advanced configuration, then we have to specify a JSON format config file to configure the hub/node and start it.

As said, hub/node can be configured in 2 different ways, one is by specifying command line parameters, and the other way is by specifying a json file which we are going to see here.

## Configuring hub using below JSON:-

{

  "port": 4444,

  "newSessionWaitTimeout": -1,

  "servlets" : [],

  "withoutServlets": [],

  "custom": {},

  "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",

  "throwOnCapabilityNotPresent": true,

  "cleanUpCycle": 5000,

  "role": "hub",

  "debug": false,

  "browserTimeout": 0,

  "timeout": 1800

}

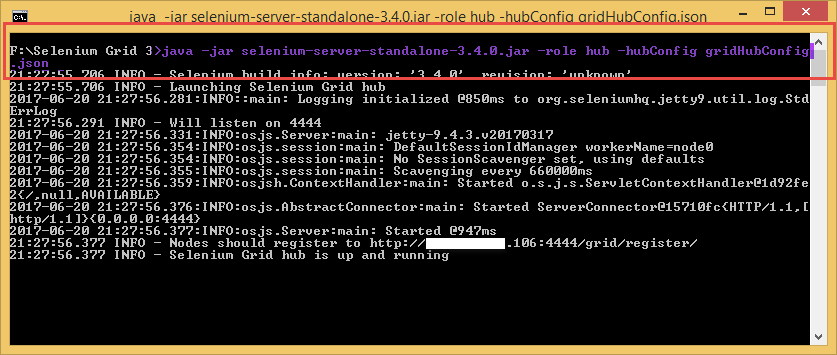
This is a sample hubconfig.json file which was made available by Selenium. You can also [download](https://github.com/SeleniumHQ/selenium/blob/master/java/server/src/org/openqa/grid/common/defaults/DefaultHub.json) this file.

## Start hub using below command :-

java -jar selenium-server-standalone-3.x.x.jar -role hub -hubConfig hubconfig.json

In our case, we used selenium latest version 3.4.0 and changed the json file name to 'gridHubConfig.json'. You can also change the json file name if needed. No other changes are made.

java -jar selenium-server-standalone-3.4.0.jar -role hub -hubConfig gridHubConfig.json



Now open your browser and enter the URL either with IP address or localhost with port number. In our case it was with - http://xx.xx.xx.106:4444/grid/console or http://localhost:4444/grid/console

You should see something like below, which indicates Grid configured successfully and is Up and running



## Configuring nodes using below JSON :-

{

  "capabilities":

  [

    {

      "browserName": "firefox",

      "maxInstances": 5,

      "seleniumProtocol": "WebDriver"

    },

    {

      "browserName": "chrome",

      "maxInstances": 5,

      "seleniumProtocol": "WebDriver"

    },

    {

      "browserName": "internet explorer",

      "maxInstances": 1,

      "seleniumProtocol": "WebDriver"

    }

  ],

  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",

  "maxSession": 5,

  "port": 5555,

  "register": true,

  "registerCycle": 5000,

  "hub": "[http://localhost:4444",](http://localhost:4444/)

  "nodeStatusCheckTimeout": 5000,

  "nodePolling": 5000,

  "role": "node",

  "unregisterIfStillDownAfter": 60000,

  "downPollingLimit": 2,

  "debug": false,

  "servlets" : [],

  "withoutServlets": [],

  "custom": {}

}

There is a sample nodeconfig.json file which was made available by Selenium. You can also [download node file.](https://github.com/SeleniumHQ/selenium/blob/master/java/server/src/org/openqa/grid/common/defaults/DefaultNodeWebDriver.json)

If you see above config file, we have not added 'host' parameter, though it is not mandatory but if you are trying to add multiple nodes from different machines, make sure to specify -remoteHost or -host with ip which you want to use.

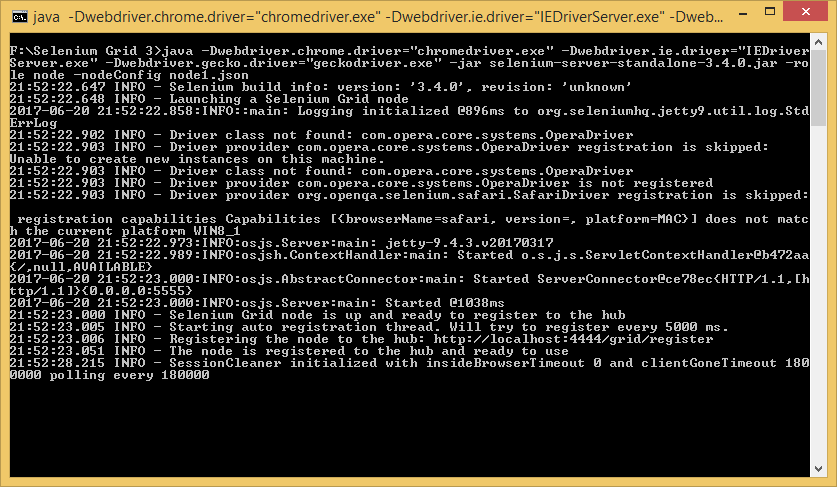
If not specified, when the node starts, it tries to guess the ip address of the host machine. If there several network interfaces, the guessing can return the wrong address and will return error as DefaultRemoteProxy unknown version, connect to :port [/] failed: Connection timed out

**NOTE:**  
If you are still using Selenium version 2.x.x, then you should download and use version 2.x.x [nodeconfig file](https://github.com/SeleniumHQ/selenium/blob/selenium-2.53.0/java/server/src/org/openqa/grid/common/defaults/DefaultNode.json" \t "_blank) as the configuration { ... } object in version 2.x.x has been removed in version 3.x.x

## Start node using below command :-

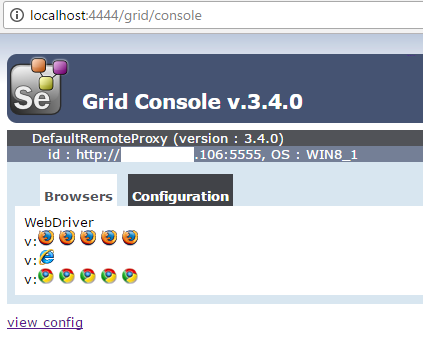
Before starting the below command, we need to make sure the driver executable are downloaded and available on node machines. As we need to set the path to driver executable while starting the nodes via command line. If the executable are in the some other directory, you have to pass ABSOLUTE\_PATH of the driver executable

java -Dwebdriver.chrome.driver="chromedriver.exe" -Dwebdriver.ie.driver="IEDriverServer.exe" -Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-3.4.0.jar -role node -nodeConfig node1.json



Now open your grid console, either with - http://xx.xx.xx.106:4444/grid/console or http://localhost:4444/grid/console

You should see something like below, which indicates node successfully registered with grid



If you just want to execute your tests on only Firefox and Chrome browser, we can remove IE capabilities from the json and then we don't have to pass IE driver executable.

java -Dwebdriver.chrome.driver="chromedriver.exe" -Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-3.4.0.jar -role node -nodeConfig node1.json

All the *-D arguments* in the command line should come before the '-jar' as Java command line usage requires system properties to be specified before the -jar. If not the you may see an exception as *"Exception in thread "main" com.beust.jcommander.ParameterException: Unknown option:"*

**Note**: If you are using selenium version 2.x.x, you don't have to pass geckodriver. Geckodriver is only needed when using selenium 3.x.x.

If you want to check list of configuration properties in detail for Selenium hub/node, run below commands

java -jar selenium-server-standalone-3.x.x.jar -role hub -help

java -jar selenium-server-standalone-3.x.x.jar -role node -help

When you are trying to launch selenium hub/node on java 1.7 using Selenium 3.x.x, you may see an exception like below, In this case you have to upgrade your JDK from Java 7 to Java 8 or higher version.  
*Exception in thread "main" java.lang.UnsupportedClassVersionError: org/openqa/grid/selenium/GridLauncherV3*