**Table Of Contents**

show

* [A Quick Walkthrough of Selenium’s Memory Lane!](https://www.interviewbit.com/blog/selenium-architecture/#a-quick-walkthrough-of-selenium%E2%80%99s-memory-lane)
* [What is Selenium?](https://www.interviewbit.com/blog/selenium-architecture/#what-is-selenium)
* [What is Selenium WebDriver?](https://www.interviewbit.com/blog/selenium-architecture/#what-is-selenium-webdriver)
* [Selenium WebDriver Architecture](https://www.interviewbit.com/blog/selenium-architecture/#selenium-webdriver-architecture)
  + [Selenium Client Library](https://www.interviewbit.com/blog/selenium-architecture/#selenium-client-library)
  + [Selenium API](https://www.interviewbit.com/blog/selenium-architecture/#selenium-api)
  + [JSON Wire protocol](https://www.interviewbit.com/blog/selenium-architecture/#json-wire-protocol)
  + [Browser Drivers](https://www.interviewbit.com/blog/selenium-architecture/#browser-drivers)
  + [Browser](https://www.interviewbit.com/blog/selenium-architecture/#browser)
* [Working of Selenium WebDriver](https://www.interviewbit.com/blog/selenium-architecture/#working-of-selenium-webdriver)
* [Advantages of Selenium WebDriver Architecture](https://www.interviewbit.com/blog/selenium-architecture/#advantages-of-selenium-webdriver-architecture)
* [Disadvantages of Selenium WebDriver](https://www.interviewbit.com/blog/selenium-architecture/#disadvantages-of-selenium-webdriver)
* [Conclusion](https://www.interviewbit.com/blog/selenium-architecture/#conclusion)

The goal of automation testing is to minimize the time and effort of testers and generate accurate test results. A tool combined with practical knowledge about the system is used to automate test execution. If you are an automation test engineer, Selenium is one tool that you would have heard about. If you want to know more about Selenium, then you have stumbled upon the right article! This article comprises crisp and concise information about Selenium Architecture, Selenium WebDriver, and its features, advantages, and disadvantages. Without further ado, let’s begin!

Selenium was developed in 2004 by Jason Huggins, a Thoughtworks engineer, to address the shortcomings of manual testing. He developed a solution in the form of a program using JavaScript. He initially named it JavaScript TestRunner. Later on, when he realized that the program could do much more, he renamed it Selenium Core and open-sourced it.  
Next Selenium Remote Control (RC) or better known as Selenium 1 was developed by Paul Hammant, another ThoughtWorks engineer. Selenium RC was developed to solve domain-related issues while testing web applications.

This further escalated to the development of Selenium Grid for parallel testing purposes. Next, Selenium IDE was developed to automate the browser through the record and playback features (similar to UFT/QTP). In 2008, the Selenium team decided to merge the web driver and Selenium RC into a tool called Selenium 2, which later evolved to Selenium 3 or better known as Selenium WebDriver.  
Selenium RC is now deprecated and moved to legacy packaging.



**What is Selenium?**

Selenium is an open-source framework used to automate the testing of web applications. If you are looking forward to automating functional and regression test cases, then Selenium would be the right choice! Test scripts can be written in Selenium using different programming languages like Java, Python, C#, Ruby, and JavaScript.  
Quick notes:  
Selenium is a web automated testing tool that supports cross-browser testing across various operating systems.  
Selenium supports JAVA, Python, C#, Ruby, and JavaScript.

**What is Selenium WebDriver?**

Selenium WebDriver is currently the most widely used component in the Selenium tool suite. Selenium WebDriver: Selenium 2 integrated with WebDriver API provides an understandable programming interface. JAVA and C# languages are mostly preferred to work with Selenium.

**Selenium WebDriver Architecture**

Selenium WebDriver is currently the most widely used component in the Selenium tool suite. Selenium WebDriver: Selenium 2 integrated with WebDriver API provides an understandable programming interface. JAVA and C# languages are mostly preferred to work with Selenium.

Before diving into Selenium WebDriver architecture, let us look at its components.  
Selenium 3’s architecture consists of five layers.

**Selenium Client Library**

Selenium Client Library or language bindings is a programming library that consists of commands in the form of an external jar file that are compatible with Selenium protocol/W3C Selenium protocol. The selenium client library can be divided into two groups:

**Web Driver protocol clients**– They are thin wrappers around WebDriver protocol HTTP calls. Based on the user’s preferred programming language, the library can be downloaded from Selenium’s official repository. The library can later be added while creating a new project or a new Maven project in Eclipse or IntelliJ.

**WebDriver-based tools** – These are higher-level libraries that allow us to work with WebDriver automation. Testing frameworks like Selenide, webdriver.io, or AI-powered Selenium extensions like Healenium come under this group. These tools rely on lower-level webdriver protocols to function efficiently.

**Selenium API**

Selenium API is a set of rules and regulations that the programs use to communicate with each other. APIs work as an interface between the program and aid in their interaction without any user knowledge.

**JSON Wire protocol**

JSON is used in web services in REST and is a widely accepted method for communication between heterogeneous systems. The Selenium WebDriver uses JSON to communicate between client libraries and drivers. The JSON requests sent by the client are converted into HTTP requests for the server’s understanding and again converted back to JSON format while sending it to the client again. This data transfer process is serialization. By this method, the internal logic of the browser is not revealed, and the server can communicate with the client libraries, even if it is not aware of any programming language.

**Browser Drivers**

Browser drivers act as a bridge between the Selenium libraries and the browsers. They help to run Selenium commands on the browser. Each of the browsers has separate drivers, which can be downloaded from Selenium’s official repository. While using a browser driver, we need to import the respective Selenium package “org.openqa.selenium.[$browsername$];” in our code. We should also set the System property of the executable file of the browser driver using the following syntax:

System.setProperty(key,value)

Where key is the driver’s name and value is the path to the executable file of the driver in the user’s device.  
Let us understand this better with this code snippet:

Package InterviewBitBlog;

import.org.openqa.selenium.chrome.ChromeDriver;

Public class IBContent{

@Test

Public void browser(){

System.setProperty("webdriver.chrome.driver", "C:\\downloads\\chromedriver.exe")

ChromeDriver driver=new ChromeDriver

When we execute this code, the chrome browser will be opened by Selenium.

**Browser**

All the browsers supported by Selenium come under this category. Selenium test scripts can be run across various browsers like Chrome, Safari, Firefox, Opera, and Internet Explorer and operating systems like Windows, Mac OS, Linux, and Solaris.  
Quick notes:  
Selenium architecture comprises 5 components; Selenium Client Library, Selenium API, JSON Wire Protocol, Browser Drivers, and Browsers.  
Selenium Client library – Selenium commands in the desired programming language in compliance with the W3C Selenium protocol.  
Selenium API – Facilitates software to software interaction.  
JSON Wire Protocol – Communication method between client libraries and drivers.  
Browser drivers – Support interaction between Selenium library and web browser.

**Working of Selenium WebDriver**

Imagine the working of Selenium WebDriver like a conversation between you, a foreign tourist, and your friend. The tourist is asking for directions, but you are new in the city. You know the tourist’s language, but your friend knows the directions in and out. So, you ask the tourist what they want in their language, and translate it for your friend. Your friend tells you the directions in a jiffy, and you quickly explain it to the tourist in their language. Sounds simple right? Now let us relate this scenario with the components in our Selenium Architecture!  
Here’s an example script for login written in Selenium JAVA (Element values used are for illustration purposes).

Package InterviewBitBlog;

import.org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.Assert;

Public class IBContent{

@BeforeTest

Public void login(){

System.setProperty("webdriver.chrome.driver", "C:\\downloads\\chromedriver.exe")

ChromeDriver driver=new ChromeDriver

@Test

public void loginAutomationTest() {

driver.get("www.samplelogin.com");

Assert.assertEquals(driver.getTitle(),"Home");

WebElement signInLink = driver.findElement(By.linkText("signin"));

signInLink.click();

WebElement user = driver.findElement(By.id("username"));

user.sendKeys("test");

WebElement pass = driver.findElement(By.id("password"));

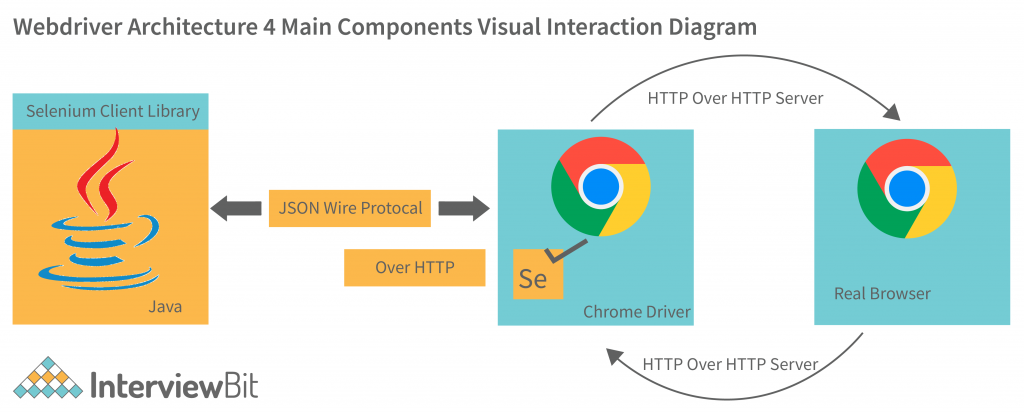
pass.sendKeys("test");

WebElement log = driver.findElement(By.name("Login"));

log.click();

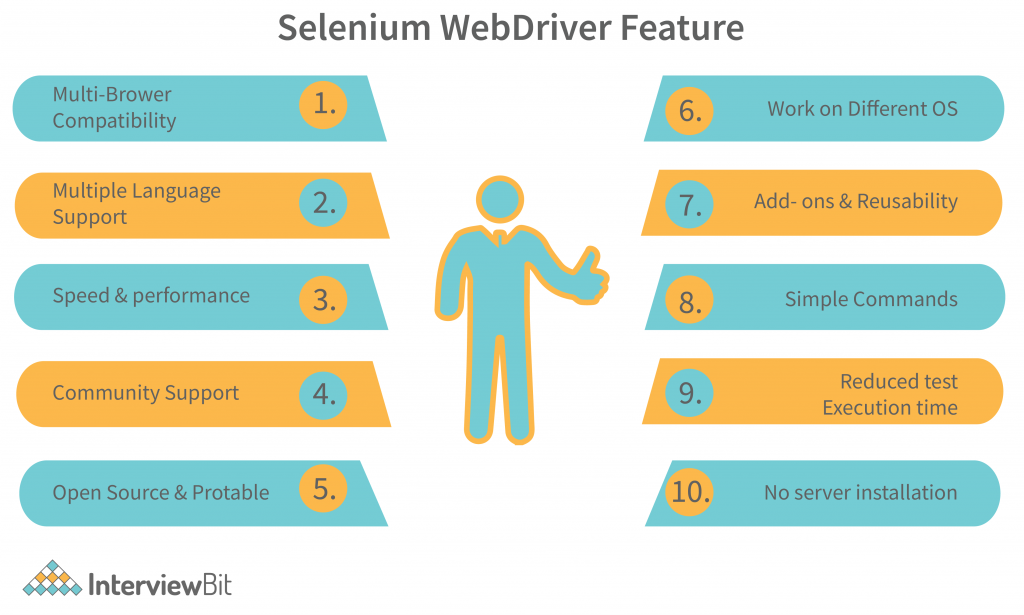
Once you have written your selenium script in the IDE, let us say, Eclipse, for now, you will hit the run button to execute the program. Based on the above program, the Chrome browser will be launched and it will navigate to “www.samplelogin.com”. The Selenium library communicates with the Selenium API, which in turn sends the programming language commands to the browser driver via the JSON wired protocol. The commands are sent in the form of JSON requests, where the protocol converts them to HTTPS requests. The browser driver will then use this HTTP server to get the request and send it to the server, where it filters out the commands that need to be executed. In this case, the driver identifies the sign-in link and performs a click operation on it. Later it identifies the username and password fields and inputs the given values and finally clicks on the “login” button. This is the execution part which is done on the browser’s UI. Finally, the HTTP server sends the response back to the test script, where the drivers and APIs convert it to JSON format and thus the results are recorded.

Here, the Selenium client library is your friend who knows the directions, the test script is the tourist and the webdriver is you. You interacted with the tourist with your multilingual skills (umm, let’s say this is the Selenium API) and successfully executed the script, that is, our tourist got their directions and reached their destination! Sounds good, doesn’t it?



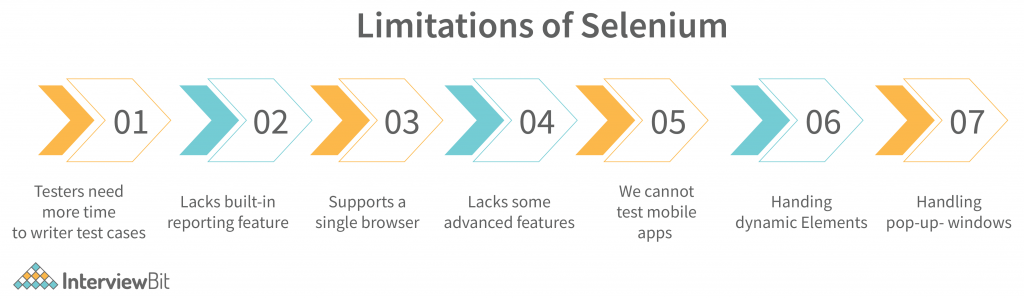
**Advantages of Selenium WebDriver Architecture**

* It is open-source, supports many languages, and is compatible with many operating systems.
* Selenium WebDriver architecture is designed to support cross-browser testing and parallel testing.
* Selenium WebDriver supports integration with various frameworks like Maven, and ANT for code compilation.
* It also supports integration with testing frameworks like TestNG to improve automation testing and reporting.
* Selenium can be integrated with Jenkins for CI/CD purposes.
* Selenium has strong community support which makes troubleshooting pretty easy.
* Selenium Architecture enables us to implement user gestures like the mouse cursor and keyboard actions like click, double click, drag, and drop, click and hold, etc.
* With Selenium, you can write your test scripts in the language with which the web application was coded, thus speeding up test cycles.
* Selenium does not require us to start any server before testing and provides a direct interpretation of code onto the web services.
* The architecture of Selenium enables us to simulate advanced browser interactions like clicking the browser’s back and front buttons.



**Disadvantages of Selenium WebDriver**

* Selenium does not support testing of Windows applications as it works only on web applications.
* Selenium depends on third-party frameworks like TestNG and Cucumber for reporting, as it does not have inbuilt reporting features.
* Selenium architecture is not prepared to handle dynamic web elements accurately, thus affecting test results.
* Selenium does not handle frames and pop-ups efficiently.
* Selenium does not automate captcha, barcodes, and test cases that involve fingerprints.
* Selenium does not support the automation of video and audio elements.
* Selenium requires knowledge of programming languages, thus making test script authoring slightly hard.
* Test management tasks cannot be performed with Selenium, while tools like UFT/QTP support ALM integration.



  [**What is Selenium?**](https://intellipaat.com/blog/selenium-architecture/#no1)

 [**What is Selenium Architecture?**](https://intellipaat.com/blog/selenium-architecture/#no2)

* [**Selenium IDE**](https://intellipaat.com/blog/selenium-architecture/#no3)
* [**Selenium RC**](https://intellipaat.com/blog/selenium-architecture/#no4)
* [**Selenium WebDriver**](https://intellipaat.com/blog/selenium-architecture/#no5)
* [**Selenium Grid**](https://intellipaat.com/blog/selenium-architecture/#no6)

 [**Why is Selenium so preferred?**](https://intellipaat.com/blog/selenium-architecture/#no7)

* [**Browser Compatibility**](https://intellipaat.com/blog/selenium-architecture/#no8)
* [**Language Compatibility**](https://intellipaat.com/blog/selenium-architecture/#no9)
* [**Prompt Implementation**](https://intellipaat.com/blog/selenium-architecture/#no10)

 [**Top Features of Selenium**](https://intellipaat.com/blog/selenium-architecture/#no11)

 [**How to use Selenium for Web Automation?**](https://intellipaat.com/blog/selenium-architecture/#no12)

 [**Conclusion**](https://intellipaat.com/blog/selenium-architecture/#no13)

## ****What is Selenium?****

[Selenium](https://intellipaat.com/blog/tutorial/selenium-tutorial/introduction/) is an automation testing tool or to be precise, a framework. It is an open-source framework that has been designed for the automation testing of web applications. Also, Selenium is a flexible testing tool that allows the automation tester to write testing scripts in Selenium in various programming languages such as Python, Java, etc.

Selenium supports various web browsers such as Safari, Firefox, Opera, and Chrome wherein the Selenium test scripts, written in different languages can be run easily. It also supports cross-platform browsing, i.e. the test cases can be run simultaneously across different platforms. The platforms supported by Selenium are Windows, Mac OS, Linux, and Solaris. Selenium has become the top automation testing tool as it allows developers to create robust and flexible automation suits.

Every day, thousands of applications are deployed to the web. Now the testing teams have to be always ready to ensure that these applications are performing up to the mark even outside the development environment. To perform this testing, a user-friendly and robust framework is needed. Selenium’s amazing suite has helped millions of app deployments easy. To understand how this has been possible, we will have to understand the architecture of Selenium.

Selenium Grid;

* [What is Selenium?](https://www.browserstack.com/guide/selenium-grid-tutorial#toc0)
* [What is Selenium Grid?](https://www.browserstack.com/guide/selenium-grid-tutorial#toc1)
* [When should testers use Selenium Grid?](https://www.browserstack.com/guide/selenium-grid-tutorial#toc2)
* [How to setup Selenium Grid for Cross Browser Testing](https://www.browserstack.com/guide/selenium-grid-tutorial#toc3)
  + [Step 1: Installation](https://www.browserstack.com/guide/selenium-grid-tutorial#toc4)
  + [Step 2: Start Hub](https://www.browserstack.com/guide/selenium-grid-tutorial#toc5)
  + [Step 3: Start Nodes](https://www.browserstack.com/guide/selenium-grid-tutorial#toc6)
  + [Step 4: Configure Nodes](https://www.browserstack.com/guide/selenium-grid-tutorial#toc7)
  + [Step 5: Using Selenium Grid to run tests](https://www.browserstack.com/guide/selenium-grid-tutorial#toc8)
* [Conclusion](https://www.browserstack.com/guide/selenium-grid-tutorial#toc9)

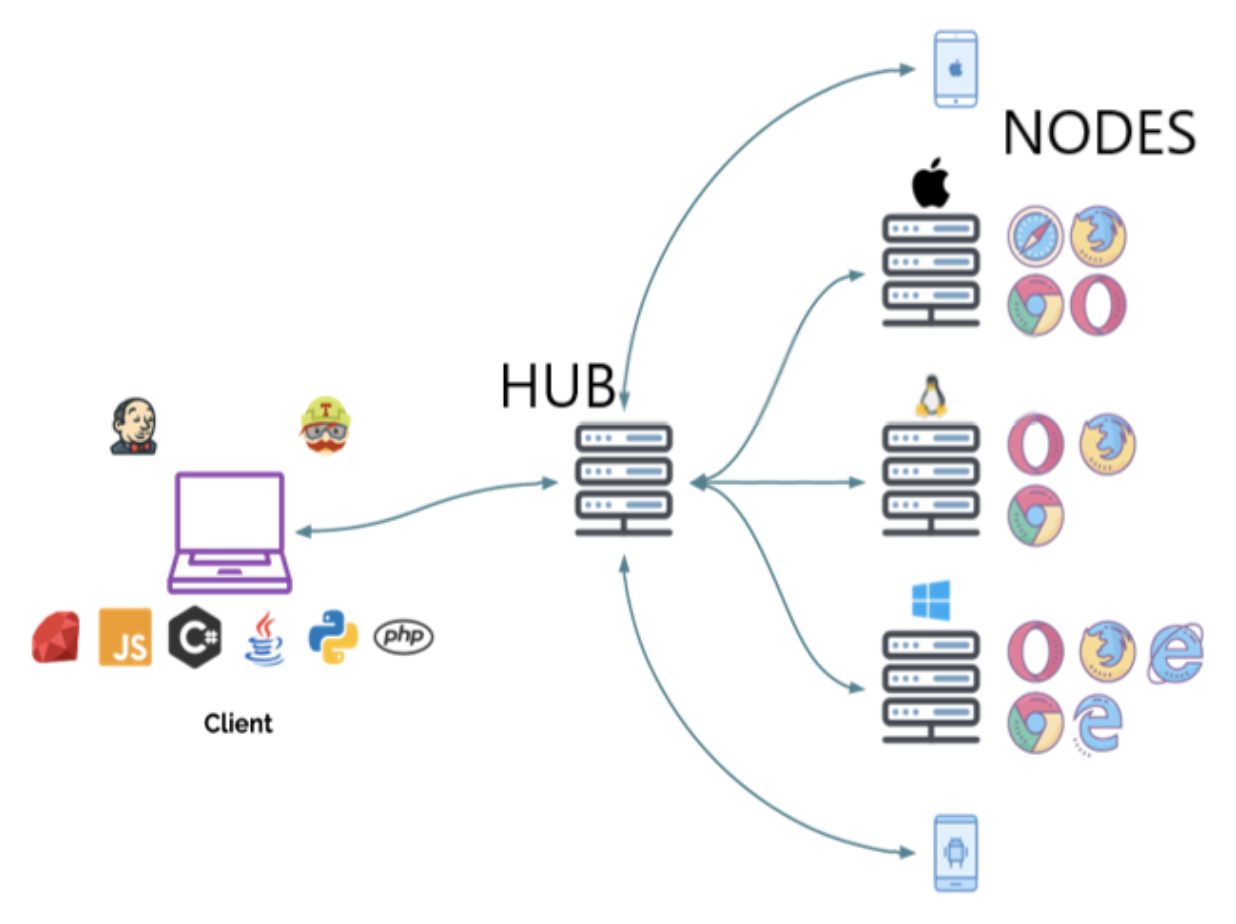
## What is Selenium Grid?

Selenium Grid is a smart proxy server that makes it easy to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Hub enables simultaneous execution of tests on multiple machines, managing different browsers centrally, instead of conducting different tests for each of them. Selenium Grid makes cross browser testing easy as a single test can be carried on multiple machines and browsers, all together, making it easy to analyze and compare the results.

The two major components of the Selenium Grid architecture are:

* **Hub** is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
* **Node** is a remote device that consists of a native OS and a remote WebDriver. It receives requests from the hub in the form of JSON test commands and executes them using WebDriver



## When should testers use Selenium Grid?

Testers should use Selenium Grid in the following circumstances:

* To run tests on multiple browsers and their versions, different devices, and operating systems
* To reduce the time that a test suite takes to complete execution

Selenium Grid improves the turnaround time of the test results. It is especially useful when the test suite is large and takes more time to run. It offers flexibility and ensures [maximum test coverage](https://www.browserstack.com/guide/how-to-ensure-test-coverage) within a limited time. Since the virtual infrastructure is in use, maintenance becomes easy.

[Run Selenium Tests for Free](https://www.browserstack.com/users/sign_up?ref=guide-selenium-grid-tutorial-bottom&product=automate)

## How to setup Selenium Grid for Cross Browser Testing

Selenium Grid can be used to perform [Cross Browser Testing](https://www.browserstack.com/cross-browser-testing) at a scale, by running a test on different browser-device combinations simultaneously. Using parallel testing, you can ensure a consistent user experience across various browser versions and devices in a short period of time. With [BrowserStack Cloud Selenium Grid](https://www.browserstack.com/cloud-selenium-grid" \o "Selenium Grid on Cloud" \t "_blank) you get access to 3000+ real device browser combinations for comprehensive cross browser testing.

To perform cross browser testing using Selenium Grid, follow the steps below for Selenium Grid configuration:

#### **Step 1: Installation**

Before getting started, [download the Selenium Server Standalone package](https://www.seleniumhq.org/download/). This package is a jar file, which includes the Hub, WebDriver, and legacy RC that is needed to run the Grid. To get started with Selenium Grid, it is essential to have Java already installed, and set up the environment variables.

#### **Step 2: Start Hub**

Hub is the central point in the Selenium Grid that routes the JSON test commands to the nodes. It receives test requests from the client and routes them to the required nodes. To set up the Selenium Hub, open the command prompt, and navigate to the directory where the Selenium Server Standalone jar file is stored (downloaded in Step 1) using the following command.

java -jar selenium-server-standalone-<version>.jar -role hub

This will start the Hub automatically using port 4444 by default. Testers can change the default port by adding an optional parameter port, using

-host <IP | hostname>

while running the command. Testers need not specify the hostname as it can be automatically determined unless someone is using an exotic network configuration or networking with VPN. In that case, specifying the host becomes necessary.

To view the status of the hub, open a browser window and navigate to <https://localhost:4444/grid/console>

#### **Step 3: Start Nodes**

Whether testers are looking to running a grid with new WebDriver functionality or with the Selenium 1 RC functionality or running both of them simultaneously, testers have to use the same [Selenium Server Standalone](https://www.browserstack.com/guide/difference-between-selenium-standalone-server-and-selenium-server) jar file, to start the nodes. To start nodes open the command prompt and navigate to the directory, where the Selenium Server Standalone jar file is stored.

Type the following command

java -jar selenium-server-standalone-<version>.jar -role node -hub https://localhost:4444/grid/register

When **-role** option that is provided is not specified, and it is not the hub, the default port is 5555. So, it is important to define the **-role** to be a node in this case.

#### **Step 4: Configure Nodes**

When testers start the nodes, by default, it allows 11 browsers, i.e., 5 Firefox, 5 Chrome, and 1 Internet Explorer for concurrent use. It also allows testers to conduct a maximum of 5 concurrent tests by default.

Testers can change this and other browser settings, by configuring nodes. This can be done by passing parameters to each of the -browser switches that represent a node, based on the parameters.

As soon as the **-browser** parameter is used, the default browser settings shall be ignored and only the parameters that are specified in the command line shall be used.

Let us understand this with an example to set 4 Firefox version 4 nodes on a Windows machine.

-browser browserName=firefox,version=4,maxInstances=4,platform=WINDOWS

In a case where the machine has [multiple versions of Firefox](https://www.browserstack.com/test-on-firefox-browser),  map the location of each binary to the compatible version on the same machine.

Let us understand this by the following example where there are two versions of Firefox, namely 3.6 and 4 on the same Windows machine that have to be used at 5 and 4 instances respectively.

-browser browserName=firefox,version=3.6,firefox\_binary=/home/myhomedir/firefox36/firefox,maxInstances=5,platform=WINDOWS -browser browserName=firefox,version=4,firefox\_binary=/home/myhomedir/firefox4/firefox,maxInstances=4,platform=WINDOWS

This way, testers can configure the nodes as per their cross browser testing requirements, using desired combination of browsers, their versions, and operating systems.

[Test on Cloud Selenium Grid for Free](https://www.browserstack.com/cloud-selenium-grid?ref=guide-selenium-grid-tutorial-mid)

#### **Step 5: Using Selenium Grid to run tests**

Once the Selenium Grid setup is done by following the above 4 steps, testers can access the grid to run tests. If Selenium 1 RC nodes are being used, testers can use **DefaultSelenium** object and pass the same in the hub formation using the following command.

Selenium selenium = new DefaultSelenium(“localhost”, 4444, “\*firefox”, “https://www.browserstack.com”);

If testers are using Remote WebDriver nodes, they must the **RemoteWebDriver** and **[DesiredCapabilities](https://www.browserstack.com/guide/desired-capabilities-in-selenium-webdriver" \o "Desired Capabilities in Selenium Webdriver" \t "_blank)** objects to define the browser, version, and platform. For this, create the target browser capabilities to run the test on:

DesiredCapabilities capability = DesiredCapabilities.firefox();

Once created, pass this set of browser capabilities into the **RemoteWebDriver** object:

WebDriver driver = new RemoteWebDriver(new URL("https://localhost:4444/wd/hub"), capability);

Once this is done, the hub would assign the test to a matching node, if all the requested capabilities meet. To request any specific capabilities on the grid, specify them before passing them to the WebDriver object in the following pattern:

capability.setBrowserName();

capability.setPlatform();

capability.setVersion()

capability.setCapability(,);

If these capabilities do not exist on the Grid, the code returns no match and thus the test fails to run.

Let us understand this using an example, considering a node is registered with the setting:

-browser browserName=firefox,version=4,maxInstances=4,platform=WINDOWS

Then, it is a match with the following set of capabilities defined for the test:

capability.setBrowserName(“firefox” );

capability.setPlatform(“WINDOWS”);

capability.setVersion(“4”);

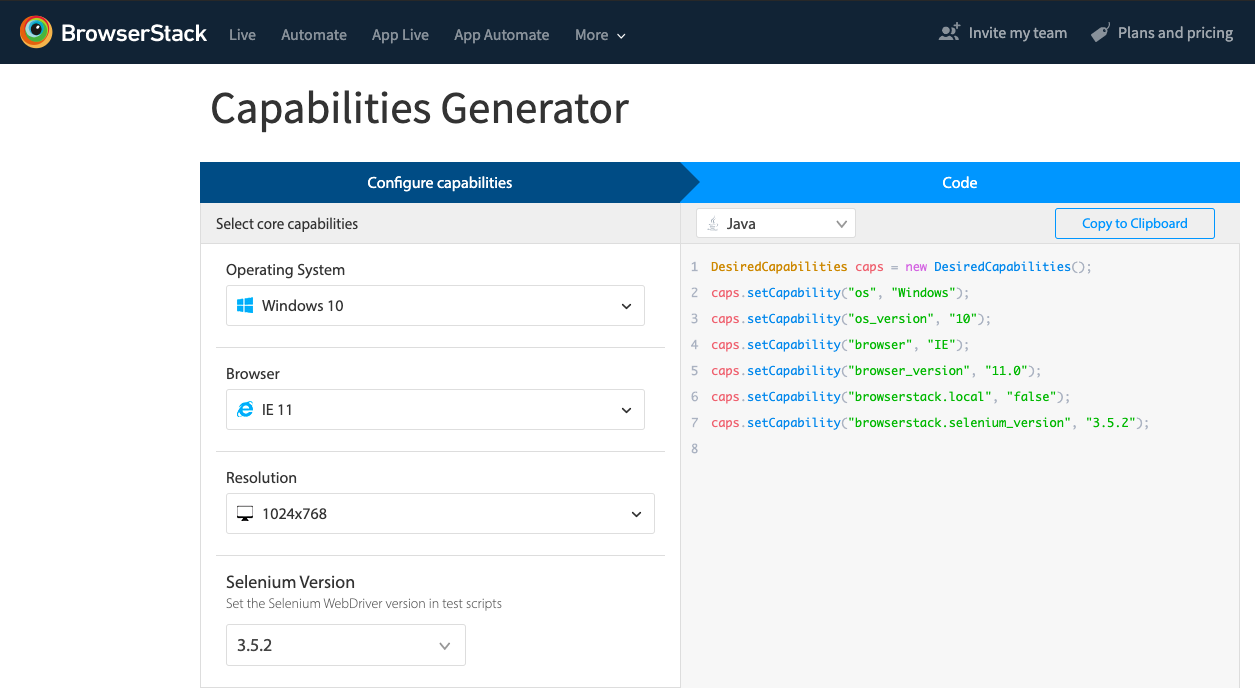
It would also match with the following set of capabilities defined for the test”

capability.setBrowserName(“firefox” );

capability.setVersion(“4”);

Note that the capabilities which are not specified for the test would be ignored, such as in the above example where the platform parameter is not specified and it gets a match.

**Note:** You can also set Desired Capabilities by using [BrowserStack Capabilities Generator](https://www.browserstack.com/automate/capabilities)



Using these steps, testers can easily set up, configure, and perform tests on Selenium Grid for concurrent execution of test suites.

**Must-Read:** [Selenium Grid 4 Tutorial](https://www.browserstack.com/guide/selenium-grid-4-tutorial)

### Conclusion

Selenium Grid offers the convenience to perform concurrent testing on several browsers, browser versions, and machines. Having learned how to use Selenium Grid, testers can ensure cross-platform compatibility of their web applications with [parallel testing](https://www.browserstack.com/guide/parallel-testing-with-selenium) on BrowserStack’s [Cloud Selenium Grid](https://www.browserstack.com/cloud-selenium-grid). The Grid allows teams to instantly access to 3000+ real browsers and devices spanning different versions, manufacturers, and operating systems. real desktops, iOS, and Android devices. This makes the testing results more accurate since you can [test under real user conditions](https://www.browserstack.com/real-user-conditions-testing-on-browserstack), hence identifying bottlenecks and rectifying them becomes easy.

[BrowserStack Automate](https://www.browserstack.com/automate) allows you too test on 3000+ real devices and browser for a comprehensive testing experience. BrowserStack [real device cloud](https://www.browserstack.com/real-device-cloud) ensure [maximum test coverage](https://www.browserstack.com/guide/how-to-ensure-test-coverage) by providing access to legacy and latest devices and browsers.

This would help deliver a consistent end-user experience that conforms with the continuous delivery approach using Selenium Grid.