# How to set up Selenium Grid for Cross Browser Testing

Selenium Grid can be used to perform 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 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. 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 run 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 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, 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/fi
refox,maxInstances=5,platform=WINDOWS -browser
browserName=firefox,version=4,firefox_binary=/home/myhomedir/firefox4/firef
ox,maxInstances=4,platform=WINDOWS
```

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

Test on Cloud Selenium Grid for Free

**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** 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:

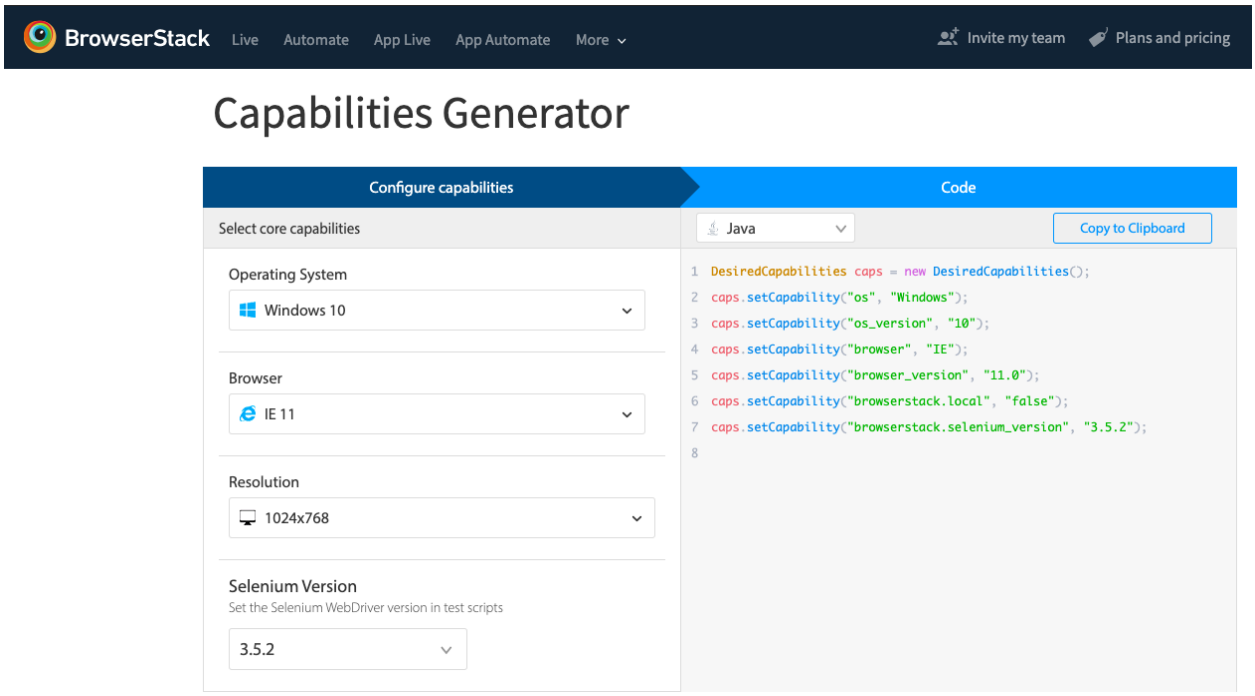
```
capability.setBrowserName("firefox" );
capability.setPlatform("WINDOWS");
capability.setVersion("4");
```

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.




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

## 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 on BrowserStack's Cloud Selenium Grid. The Grid allows teams to instantly access 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, hence identifying bottlenecks and rectifying them becomes easy.

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