Verification of your web-application/web product with the ‘local’ version of the Selenium WebDriver is not a scalable approach. Some of the problems that you encounter with this kind of setup are:

* How to verify functionalities on platforms/operating systems which are not installed on the machine? i.e. even if it is a dual-boot machine, you will still have the problem.
* How to perform automated cross browser testing on browsers with different versions? e.g. your machine can have Firefox version 64.0, but you want to perform the testing on Firefox version 36.0.
* How to perform cross browser testing of a website on multiple number of browsers? e.g. your machine may not have Opera installed but for the purpose of testing, you would need to install these browsers (unnecessary disk space consumption).
* How to attain the best performance i.e. parallelization while performing the tests on local machine?

We will be making use of the following languages, frameworks & tools (IDE) to write better automation code.

* **Programming Language** – Python ([Download Link](https://www.python.org/downloads/))
* **Test framework** – pytest
* **Web application framework** – Selenium
* **Integrated Development Environment** – Eclipse version 4.90 ([Download Link](https://www.eclipse.org/downloads/)), PyCharm – Community Edition ([Download Link](https://www.jetbrains.com/pycharm/download/))

## What Is Selenium Grid?

Selenium Grid is a testing tool (which is a part of the Selenium Suite) that is based on the ‘client-server’ architecture. In Selenium Grid terminology, Client machine is termed as ‘Hub’ and server(s) are termed as ‘Nodes’.

Selenium Grid setup allows you to execute cross browser testing through a variety of different machines across different browsers (as well as different versions of browsers) & different operating systems. Hence, it brings the require amount of ‘parallelism’ & ‘distribution’ to your test execution environment.

A Selenium Grid setup can have only one Hub and ‘n’ number of nodes. The primary job of the ‘hub machine’ is to distribute the test case supplied to the ‘node machine’ which matches the capabilities/requirements required for executing the test case for performing cross browser testing. We would discuss more Hub & Node (which are the core components for the setup of Selenium Grid infrastructure) in further sections.

There are two versions of Selenium Grid available, namely Selenium Grid 2.0 & Selenium Grid 1.0. Selenium Grid 2.0 is most popular amongst automation testers since it supports Selenium RC (Remote Control) and Selenium WebDriver scripts.

## Why Selenium Grid Setup Is A Smart Call To Make?

There are a number of benefits of Selenium Grid setup. Here, we have a look at some of the top advantages of Selenium Grid setup as a part of your test execution strategy for performing cross browser testing.

* **Reduced Execution Time** – Selenium Grid setup can be used to execute multiple test cases on different browsers (and browser versions) and operating systems. You can also run multiple instances of Selenium RC along with the test configuring combinations. As the tests are executed in ‘parallel’ (scattered across different machines), the overall time consumed for cross browser testing trims down significantly.
* **Scalable & Maintainable** – As we know, Selenium Grid setup reduces the amount of time required for ‘parallel testing’ by a huge margin, the entire solution is highly scalable. If you want to perform testing on a ‘new node’ (which could be Operating System/Browser), all it needs are few minor tweaks in the code with respect to the addition of capabilities, and your solution for the new node is ready.
* **Perform Multi-Combination Testing** – Using the Hub & Node feature of Selenium Grid setup, you can perform multi-OS, multi-browser, and multi-device verification of your source code.

By default, the execution after the setup of Selenium Grid is not parallel in nature. In order to take the advantage of parallelism, the selenium tests should be written in a manner that takes ‘parallel execution’ for cross browser testing into account.

## Primary Components Of The Selenium Grid Setup & Brief Look At RemoteWebDriver Workflow

In our previous articles on tutorial of Selenium webdriver for cross browser testing, the core focus was on ‘Selenium for local testing’, you would have observed that a local WebDriver API/interface has to be used in order to perform the testing. In case you want to use the Selenium WebDriver to execute tests on a separate machine, you have to use Remote WebDriver interface. Since the tests (in normal scenarios) have to be executed on a different machine, **RemoteWebDriver** is based on the traditional ‘Client-Server’ model. The RemoteWebDriver consists of two main parts – **Hub (Server)** and **Node(Client)**. Let’s have a detailed look at the same in this section.

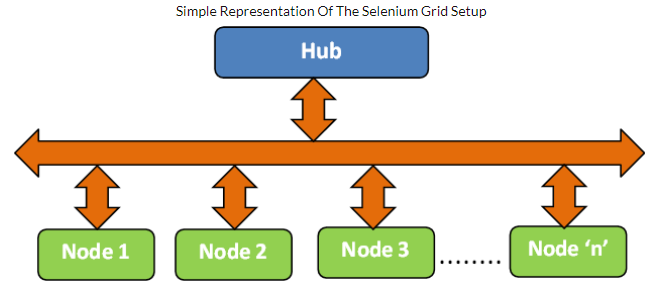
**Hub/Server** – The Hub is the central component of the ‘Selenium Grid architecture’. It loads the tests that have to be executed. There can only be one ‘Hub’ which acts as the ‘Master’ in the setup of Selenium Grid infrastructure. Once the Hub receives an input/test case that needs to be executed, it searches for a Node (client) which matches the **desired capabilities** and diverts the ‘test execution request’ to the matching node.

For example, if you have setup a Selenium Grid with (hub + two nodes) where the node configuration is as below:

* Node 1 – Windows 8.1, Chrome 73.0
* Node 2 – Windows 10, Firefox 64.0

If there is a test case that has the ‘Desired Capabilities requirement’ – (Windows 10 + Chrome), the hub first receives the request for this requirement. By looking at the requirement, it diverts the execution request to Node 1.

**Node** – Node is a machine on which the tests are executed. Node can have a different configuration than the hub. There is no limit on the number of nodes that can be connected to the hub. Since the test code execution will occur on the node(s); it is recommended that you choose machines with the best possible configuration so that you can get the best performance out of your Selenium Grid setup.



## Installation & Configuration To Setup A Selenium Grid

[**Note** – Our development environment is installed on Windows 10 machine. For demonstration, Hub & Node are installed on the same machine.]

**Step 1** – In order to setup Selenium Grid infrastructure, you need to first download the Selenium Server jar from the official website of Selenium. It was formerly called as the ‘Selenium RC server’. The download link is [here](https://www.seleniumhq.org/download/).

Once the selenium server jar file is downloaded, you should configure the hub using following command (java -jar selenium-server-standalone-x.x.x.jar -role hub)

java -jar selenium-server-standalone-3.141.0.jar -role hub

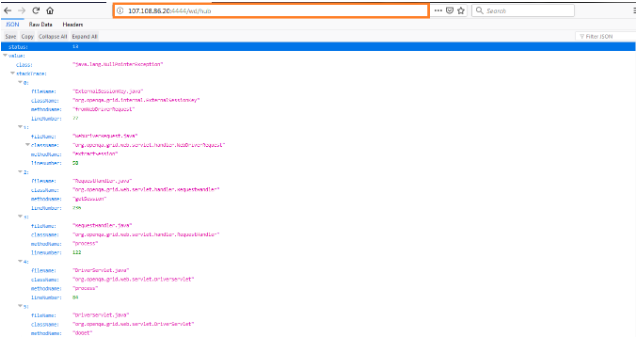
**Step 2** – By default, the hub would use the port 4444. The port can be changed by using the configuration option –port . Below is the screenshot of the logs observed once the above command is executed.



In order to verify whether the hub is configured correctly, open the browser window and type the following in the address bar:

http://192.168.0.194:4444/wd/hub

Below is the output snapshot when the above URL is opened in the Firefox browser:



**Step 3** – Now that the ‘Hub’ is up & running, you have to now configure the nodes for the purpose of cross browser testing. Execute the following command on the terminal:

java -jar selenium-server-standalone-3.141.0.jar -role node -hub http:///grid/register/ -browser "browserName=firefox,maxinstance=1,platform=WINDOWS" –port

In our case, the Hub is configured on port number 4444 and the node is configured on port numbers 5555 & 5556. One node is configured for Platform as Windows & browser as Firefox. Second node is configured for Platform as Windows & browser as Chrome.

java -jar selenium-server-standalone-3.141.0.jar -role node -hub http://192.168.0.194:4444/grid/register/ -browser "browserName=firefox,maxinstance=1,platform=WINDOWS" -port 5555

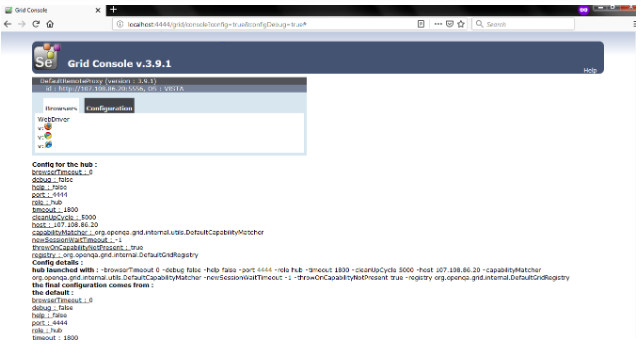
java -jar selenium-server-standalone-3.141.0.jar -role node -hub http://192.168.0.194:4444/grid/register/ -browser "browserName=chrome,maxinstance=1,platform=WINDOWS" -port 5556

The output for the command is as shown below.



For verifying whether the Node has been configured correctly, please enter the following in the address bar.

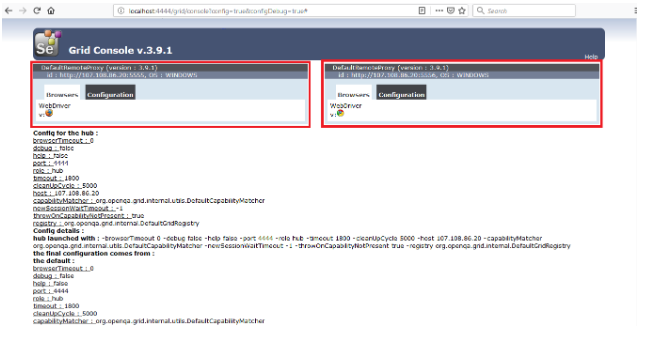
http://localhost:4444/grid/console?config=true&configDebug=true#



For configuring the node on port number 5556 (with Operating System as Windows and Browser as Chrome), please execute the following command on the terminal:

java -jar selenium-server-standalone-3.9.1.jar -role node -hub http://107.108.86.20:4444/grid/register/ -port 5556 -browser "browserName=chrome,maxinstance=1,platform=WINDOWS"

Now, when you visit http://localhost:4444/grid/console?config=true&configDebug=true# on the address bar, you will also see details about ports 5556 as well.



In case you are planning to use the Internet Explorer browser, you need to just pass the correct browser name as an input argument to the jar file. Below is the command which you can use for using Internet Explorer browser:

java -jar selenium-server-standalone-3.9.1.jar -role node -hub http://107.108.86.20:4444/grid/register/ -port 5556 -browser "browserName=internet explorer,maxinstance=1,platform=WINDOWS"

There are other parameters apart from browserName which you might require, if needed:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **CONFIGURATION OPTION** | **EXPLANATION** | | maxinstance | Used to put a restriction on the maximum number of browser initializations in the code. In case you want maximum 3 Firefox instances on the node 5555, you can configure using the option -browser “browserName=firefox,maxinstance=3,platform=WINDOWS” -port 5555 | | maxSession | Configure the number of browser instances that can run in parallel on the node. Requests are automatically queued once the maxSession limit is reached. In order to limit the number of Firefox sessions on port 5555, you can use the following option -browser “browserName=firefox,maxinstance=1,platform=WINDOWS” -maxSession 3 -port 5555 | |
| |  |  | | --- | --- | | port | Configure the port number to which the node will be connected. Configured using –port option. | |
| |  |  | | --- | --- | | maxinstance | Used to put a restriction on the maximum number of browser initializations in the code. In case you want maximum 3 Firefox instances on the node 5555, you can configure using the option -browser “browserName=firefox,maxinstance=3,platform=WINDOWS” -port 5555 | |
| |  |  | | --- | --- | | maxSession | Configure the number of browser instances that can run in parallel on the node. Requests are automatically queued once the maxSession limit is reached. In order to limit the number of Firefox sessions on port 5555, you can use the following option -browser “browserName=firefox,maxinstance=1,platform=WINDOWS” -maxSession 3 -port 5555 | |

## Using Pytest With Your Selenium Grid Setup

Now that you have knowledge about Selenium Grid, let’s have a look at an example where we use pytest with our Selenium Grid setup(or the Remote WebDriver interface). We have a detailed article about test automation using Pytest and Selenium WebDriver interface where we have explained about the installation of pytest, Selenium with pytest, fixtures in pytest, etc.

For the purpose of demonstration, Hub (Server) & Nodes (Clients) are configured on the same machine. We would be using the machine IP address when configuring the Remote WebDriver. Let’s configure the Hub & Nodes using the respective commands for automated cross browser testing.

### **Setting Up The Hub For Your Selenium Grid**

Please go to the directory where you have the Selenium Server jar file (In our case, we are using the version 3.9.1 and the jar file name is selenium-server-standalone-3.9.1.jar) by using the ‘cd’ command. Start the Hub by executing the below command on the terminal:

java -jar selenium-server-standalone-3.9.1.jar -role hub

As discussed before, by default the Hub would start on port 4444. ‘Nodes’ can connect to the ‘Hub’ using the link **http://:/wd/hub**. In our case, the Hub/Server is configured on a machine that has IP address ‘107.108.86.20’ and the port on which it is configured is 4444. Hence, nodes need to use the link **http://107.108.86.20:4444/wd/hub** in order to divert their request to the configured Hub.

### **Setting Up The Nodes For Your Selenium Grid**

Now that the Hub is configured, the next step for setup of Selenium Grid is the Nodes. We will configure two nodes that have the below requirements.

* **Node 1** – Port 5555, Firefox, maxinstance = 1, platform = Windows
* **Node 2** – Port 5556, Chrome, maxinstance = 1, platform = Windows

On the terminal, execute the below commands to configure the two nodes:

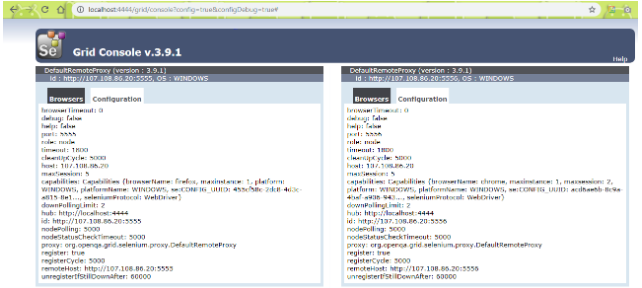
**Configuration of Node 1**

java -jar selenium-server-standalone-3.9.1.jar -role node -hub http://107.108.86.20:4444/grid/register/ -browser "browserName=firefox,maxinstance=1,platform=WINDOWS" -port 5555

**Configuration of Node 2**

**java -jar selenium-server-standalone-3.9.1.jar -role node -hub http://107.108.86.20:4444/grid/register/ -browser "browserName=chrome,maxinstance=1,platform=WINDOWS" -port 5556**

You can visit to verify whether the nodes have been configured properly. Below is the screenshot which shows the status of the nodes.



The end goal that we plan to achieve is to divert the request from the nodes to the hub, invoke the ‘required browser instance’, open the webpage inputted to the browser & terminate the browser session i.e. perform cleaning up of allocated resources.

Since we are using pytest for development, we recommend that you download PyCharm (Community Edition) from [here](https://www.jetbrains.com/pycharm/download/).

**command\_executor** – Information about the Hub i.e. the address on which the Hub is configured. If the hub is configured incorrectly i.e. wrong IP address/wrong port is used, you may get an error ‘Couldn’t register the node’ while doing the node registration.



**desired\_capabilities** – The required capabilities/requirements are passed through the Remote WebDriver. Based on the details inputted as a part of desired\_capabilities, the ‘best matching’ node is selected for execution.



The first test case has to be executed on a node that has ‘Chrome browser’ and the second test case requires has to be executed on a node that has ‘Firefox browser’. You can have a look at how we modified ‘Local WebDriver’ implementation to ‘Remote WebDriver’ implementation by configuring the WebDriver to execute requests on a remote machine i.e. matching node. The complete implementation is below:

#FileName - test\_selenium\_grid\_firefox\_chrome.py

# Import the 'modules' that are required for the execution

import pytest

#import pytest\_html

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.keys import Keys

from time import sleep

#Fixture for Firefox

@pytest.fixture(params=["chrome", "firefox"],scope="class")

def driver\_init(request):

if request.param == "chrome":

#Local webdriver implementation

#web\_driver = webdriver.Chrome()

#Remote WebDriver implementation

web\_driver = webdriver.Remote(

command\_executor='http://107.108.86.20:4444/wd/hub',

desired\_capabilities={'browserName': 'chrome', 'javascriptEnabled': True})

if request.param == "firefox":

#Local webdriver implementation

#web\_driver = webdriver.Firefox()

#Remote WebDriver implementation

web\_driver = webdriver.Remote(

command\_executor='http://107.108.86.20:4444/wd/hub',

desired\_capabilities={'browserName': 'firefox'})

request.cls.driver = web\_driver

yield

web\_driver.close()

@pytest.mark.usefixtures("driver\_init")

class BasicTest:

pass

class Test\_URL(BasicTest):

def test\_open\_url(self):

self.driver.get("https://www.lambdatest.com/")

print(self.driver.title)

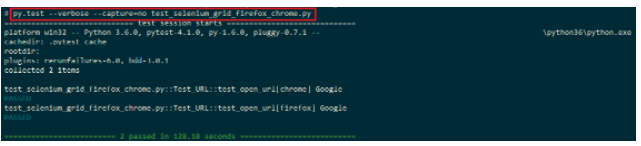
sleep(5)

The remaining implementation remains the same since the only thing that changes is using Remote WebDriver interface.

To execute the code, please use the command on the local terminal (or terminal on PyCharm).

py.test --verbose --capture=no test\_selenium\_grid\_firefox\_chrome.py

Once the Hub receives a request for the execution of ‘testcase 1’, the request would be diverted to Node 2 (where Chrome browser is configured) and request for the execution of ‘testcase 2’, the request would be diverted to Node 1 (where Firefox browser is configured). The output is shown below.



capabilities = {

"build" : "your build name",

"name" : "your test name",

"platform" : "Windows 10",

"browserName" : "Firefox",

"version" : "64.0",

"selenium\_version" : "3.13.0",

"firefox.driver" : v0.23.0

}