**Selenium Grid**

**Why Selenium Grid?**

The Selenium Grid is used because of many reasons. Here are a few

* When we want to run our tests against multiple browsers, the multiple versions of browsers and the browsers running on different operating system.
* It is also used to reduce the time taken by the test suite to complete a test pass by running tests in parallel.

**What is Selenium Grid?**

The Selenium Grid is a testing tool which allows us to run our tests on different machines against different browsers. It is a part of the Selenium Suite which specialize in running multiple tests across different browsers, operating system, and machines. You can connect to it with Selenium Remote by specifying the browser, browser version, and operating system you want. You specify these values through Selenium Remote's Capabilities.

With Selenium Grid you can create a network of connected test machines (also called nodes). This network of test machines is controlled by a Hub, using which you can run your tests on different connected nodes. Each node is basically a computer (even a virtual machine) with a combination of Operating system and Browsers. This enables us to create a network of test machines with varying combinations of Operating system and browsers. Using Selenium Grid you can run tests on a variety of Operating System and Browser combinations.

There are two main elements to Selenium Grid -- a hub, and nodes.

**What is a Hub?**

In Selenium Grid, the hub is a computer which is the central point where we can load our tests into. Hub also acts as a server because of which it acts as a central point to control the network of Test machines. The Selenium Grid has only one hub and it is the master of the network. When a test with given DesiredCapabilities is given to Hub, the Hub searches for the node which matches the given configuration. For example, you can say that you want to run the test on Windows 10 and on Chrome browser with version XXX. Hub will try to find a machine in the Grid which matches the criterion and will run the test on that Machine. If there is no match, then hub returns an error. There should be only one hub in a Grid.

**What is a Node?**

In Selenium Grid, a node is referred to a Test Machine which opts to connect with the Hub. This test machine will be used by Hub to run tests on. A Grid network can have multiple nodes. A node is supposed to have different platforms i.e. different operating system and browsers. The node does not need the same platform for running as that of hub.

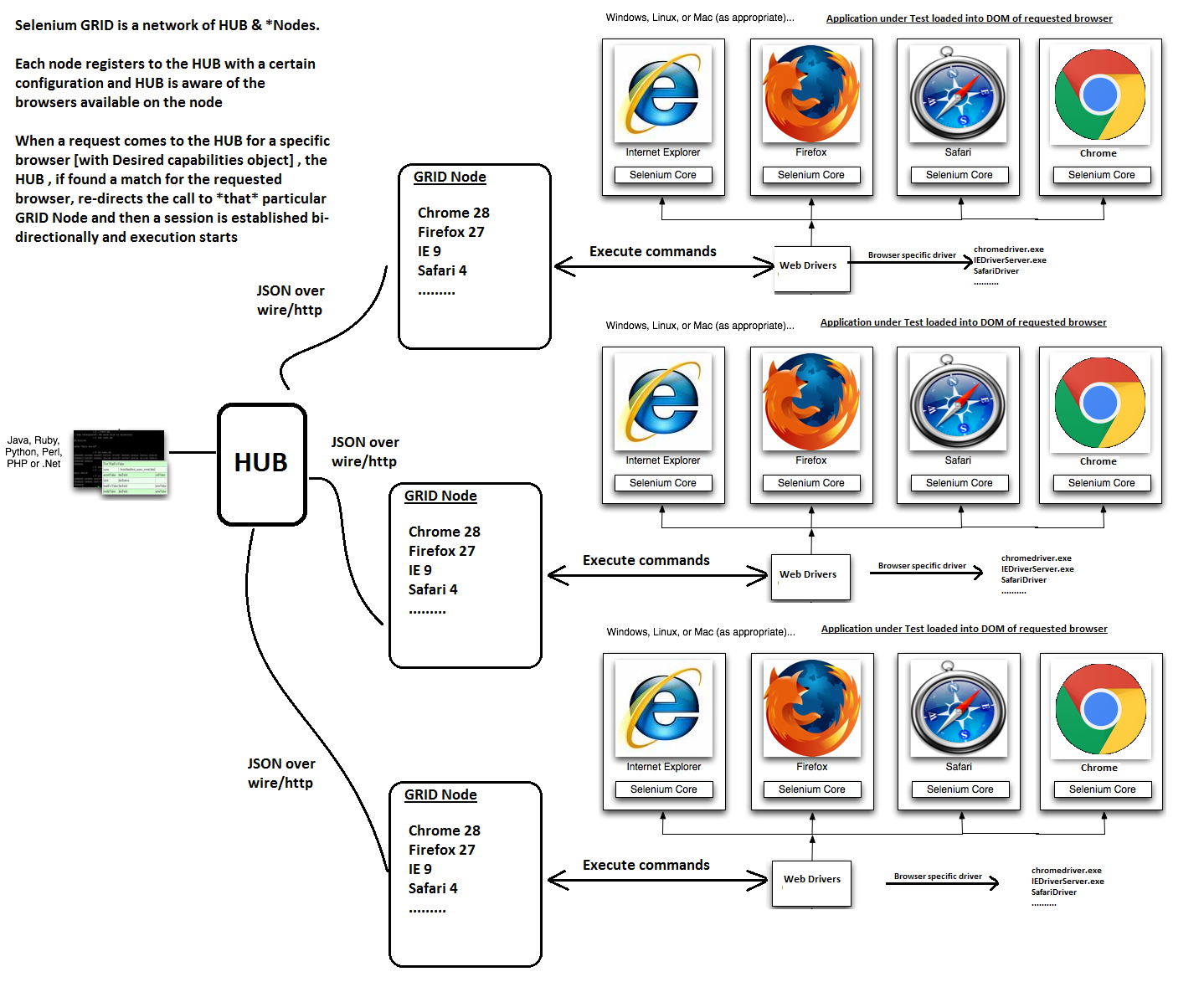
**How it works?**

First you need to create a hub. Then you can connect (or "register") nodes to that hub. Nodes are where your tests will run, and the hub is responsible for making sure your tests end up on the right one (e.g., the machine with the operating system and browser you specified in your test).

**Architecture and RemoteWebDriver Workflow**

You can use RemoteWebDriver the same way you would use WebDriver locally. The primary difference is that RemoteWebDriver needs to be configured so that it can run your tests on a separate machine. The RemoteWebDriver is composed of two pieces: a client and a server. The client is your WebDriver test, and the server is simply a Java servlet, which can be hosted in any modern JEE app server.

* RemoteWebDriver is an implementation class of the WebDriver interface that a test script developer can use to execute their test scripts via the RemoteWebDriver server on a remote machine.
* There are two parts to RemoteWebDriver: a server(hub) and a client(node)
  + The RemoteWebDriverserver is a component that listens on a port for various requests from a RemoteWebDriver Once it receives the requests, it forwards them to any of the following: Firefox Driver, IE Driver, or Chrome Driver, whichever is asked.
  + The language-binding client libraries that serve as a RemoteWebDriver The client, as it used to when executing tests locally, translates your test script requests to JSON payload and sends them across to the RemoteWebDriverserver using the JSON wire protocol.
* When you execute your tests locally, the WebDriver client libraries talk to your Firefox Driver, IE Driver, or Chrome Driver directly. Now, when you try to execute your tests remotely, the WebDriver client libraries talk to the RemoteWebDriverserver and the server talks to either the Firefox Driver, IE Driver, or Chrome Driver, whichever the WebDriver client asks for.



We will do this step by step as we move in this tutorial, but first let’s look at the overall architecture diagram as to how the workflow happens. Some jargon below.

* **JSON over wire**: JSON messages hold the entire information that is required by the server [can be Selenium GRID or WebDriver components] and hence instead of RPC, plain JSON messages are exchanged between server and client. Also referred to as WebDriver protocol
* **Desired Capabilities**: Object in automation code that has requesting browser configuration information
* **RemoteWebDriver**: Object in automation code that knows how to communicate remotely with WebDriver / Selenium GRID

**Working with Grid**

* Configuring the Hub
* Configuring the Nodes
* Develop the Script and Prepare the XML File
* Test Execution
* Result Analysis

**NOTE :** 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

**Configuring the Hub by specifying command line parameters**

**Step 1** − Download the latest Selenium Server standalone JAR file from <http://docs.seleniumhq.org/download/>.

**Step 2** − Start the Hub by launching the Selenium Server using the following command.

**Note**: Default grid port is 4444

**Note** − Ensure that there are no other applications that are running on port# 4444.

java -jar selenium-server-standalone-2.25.0.jar -role hub

Now we will use other port like '4455' to start the hub.

java -jar selenium-server-standalone-2.25.0.jar -port 4455 -role hub

**Step 3** − Now open the browser and navigate to the URL http//localhost:port or systemip:port from the Hub (The system where you have executed Step#2).

**Step 4** − Now click on the 'console' link and click 'view config'. The config of the hub would be displayed as follows. As of now, we haven't got any nodes, hence we will not be able to see the details.

**Configuring the Nodes by specifying command line parameters**

**Firefox:**

**Step 1** − Logon to the node (where you would like to execute the scripts) and place the 'selenium-server-standalone-2.42.2' in a folder. We need to point to the selenium-server-standalone JAR while launching the nodes.

**Step 2** − Launch Firefox Node using the following below command.

java -Dwebdriver.gecko.driver=D:\ geckodriver.exe -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role node -hub http://10.30.217.157:4444/grid/register -browser browserName = firefox -port 5555

Where,

D:\JAR\selenium-server-standalone-2.42.2.jar = Location of the Selenium Server Standalone Jar File(on the Node Machine)

http://10.30.217.157:4444 = IP Address of the Hub and 4444 is the port of the Hub

browserName = firefox (Parameter to specify the Browser name on Nodes)

5555 = Port on which Firefox Node would be up and running.

**Step 3** − After executing the command, come back to the Hub. Navigate to the URL - http://10.30.217.157:4444 and the Hub would now display the node attached to it.

**IE :**

**Step 1** − Now let us launch the Internet Explorer Node. For launching the IE Node, we need to have the Internet Explorer driver downloaded on the node machine.

**Step 2** − To download the Internet Explorer driver, navigate to <http://docs.seleniumhq.org/download/> and download the appropriate file based on the architecture of your OS. After you have downloaded, unzip the exe file and place in it a folder which has to be referred while launching IE nodes.

**Step 3** − Launch IE using the following command.

C:\>java -Dwebdriver.ie.driver = D:\IEDriverServer.exe

-jar D:\JAR\selenium-server-standalone-2.42.2.jar

-role node -hub http://10.30.217.157:4444/grid/register

-browser browserName = ie,platform = WINDOWS -port 5558

Where,

D:\IEDriverServer.exe = The location of the downloaded the IE Driver(on the Node Machine)

D:\JAR\selenium-server-standalone-2.42.2.jar = Location of the Selenium Server Standalone Jar File(on the Node Machine)

http://10.30.217.157:4444 = IP Address of the Hub and 4444 is the port of the Hub

browserName = ie (Parameter to specify the Browser name on Nodes)

5558 = Port on which IE Node would be up and running.

**Step 4** − After executing the command, come back to the Hub. Navigate to the URL - http://10.30.217.157:4444 and the Hub would now display the IE node attached to it.

**CHROME:**

**Step 1** − Let us now launch Chrome Node. For launching the Chrome Node, we need to have the Chrome driver downloaded on the node machine.

**Step 2** − To download the Chrome Driver, navigate to <http://docs.seleniumhq.org/download/> and then navigate to Third Party Browser Drivers area and click on the version number '2.10' as shown below.

**Step 3** − Download the driver based on the type of your OS. We will execute it on Windows environment; hence we will download the Windows Chrome Driver. After you have downloaded, unzip the exe file, and place it in a folder which has to be referred while launching chrome nodes.

**Step 4** − Launch Chrome using the following command.

C:\>java -Dwebdriver.chrome.driver = D:\chromedriver.exe

-jar D:\JAR\selenium-server-standalone-2.42.2.jar

-role node -hub http://10.30.217.157:4444/grid/register

-browser browserName = chrome, platform = WINDOWS -port 5557

Where,

D:\chromedriver.exe = The location of the downloaded the chrome Driver(on the Node Machine)

D:\JAR\selenium-server-standalone-2.42.2.jar = Location of the Selenium Server Standalone Jar File(on the Node Machine)

http://10.30.217.157:4444 = IP Address of the Hub and 4444 is the port of the Hub

browserName = chrome (Parameter to specify the Browser name on Nodes)

5557 = Port on which chrome Node would be up and running.

**Step 5** − After executing the command, come back to the Hub. Navigate to the URL - http://10.30.217.157:4444 and the Hub would now display the chrome node attached to it.

**Configuring Multiple browsers by specifying command line parameters :**

java -Dwebdriver.chrome.driver=D:\Selenium\_Practices\chromedriver\chromedriver.exe -Dwebdriver.gecko.driver=D:\Selenium\_Practices\chromedriver\geckodriver.exe -Dwebdriver.ie.driver=D:\\Selenium\_Practices\\chromedriver\\IEDriverServer.exe -jar selenium-server-standalone-3.9.1.jar -role node -hub http://localhost:4999/grid/register -browser "browserName=chrome,version=66.0,maxInstances=5,seleniumProtocol=WebDriver" -browser "browserName=firefox,version=60.0.1,maxInstances=5,seleniumProtocol=WebDriver" -browser "browserName=internet explorer,version=10.0,maxInstances=5,seleniumProtocol=WebDriver" -port 4489

**Configuring hub using JSON:**

**HUB Configuration:**

**Hub1.json file:**

{

"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

}

**Cmd syntax:**

java -jar selenium-server-standalone-3.9.1.jar -role hub -hubConfig hub1.json

**Node1.json file:**

{

"capabilities":

[

{

"browserName": "chrome",

"platform": "WINDOWS",

"maxInstances": 5,

"seleniumProtocol": "WebDriver",

"webdriver.chrome.driver": "D:/Selenium\_Practices/chromedriver/chromedriver.exe",

"binary":"C:/Program Files (x86)/Google/Chrome/Application/chrome.exe"

},

{

"browserName": "firefox",

"platform": "WINDOWS",

"maxInstances": 5,

"seleniumProtocol": "WebDriver",

"webdriver.gecko.driver": "D:/Selenium\_Practices/chromedriver/geckodriver.exe",

"binary":"C:/Program Files/Mozilla Firefox/firefox.exe"

}

],

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

"maxSession": 5,

"port": 5555,

"register": true,

"registerCycle": 5000,

"hub": "http://localhost:4444",

"nodeStatusCheckTimeout": 5000,

"nodePolling": 5000,

"role": "node",

"unregisterIfStillDownAfter": 60000,

"downPollingLimit": 2,

"debug": false,

"servlets" : [],

"withoutServlets": [],

"custom": {}

}

**Cmd syntax:**

java -jar selenium-server-standalone-3.9.1.jar -role node -nodeConfig Node1.json