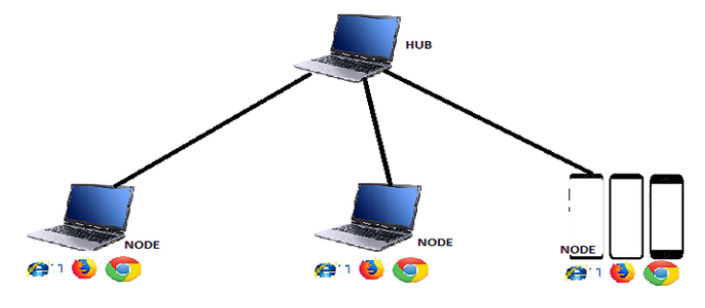
**Selenium Grid**



The **Selenium Grid** is a testing tool which allows us to run our tests on different machines against different browsers.

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

**Hub:** 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.

Nodes: 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.

### **Install Selenium GRID**

**Step 1**: Download Selenium Server jar file from Selenium’s official website which is formerly known as Selenium RC Server and save it at any location on the local disk.

URL of selenium HQ: <http://www.seleniumhq.org/download/>

**Step 2**: Start the Hub by launching the Selenium Server using the following command. Now we will use the port '4444' to start the hub.

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

java -jar selenium-server-standalone-2.25.0.jar -port 4444 -role hub -nodeTimeout 1000

**Step 3** − Now open the browser and navigate to the URL http//localhost:4444 from the Hub (The system where you have executed Step#2).

### **Configuring the Nodes**

****

**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 and Inmar Chrome Nodes using the following command with port 5556 and 5557

java -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 5556

java -Dwebdriver.chrome.driver=D:\chromedriver.exe -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role webdriver -hub http://10.30.217.157:4444/grid/register -browser browserName=chrome,platform=WINDOWS -port 5557

**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 chrome node attached to it.

### **TESTING:**

Our test automated script will test Inmar.com landing page

### **Test Dependencies**

* IDE - I'll use Intellij IDE
* Maven
* JAVA  
  The test will be written using JAVA programming language.

### **Create Test Project On IntelliJ**

* Launch IntelliJ IDE
* Click on file, select New and click on project
* On New Project dialog box, Select Maven and click on Next button
* Fill in the Group Id such as test\_id, ArtifactId such as testid and click next.
* Specify the project name (Inmar) and click finish.

Now, let's add the following plugins to the project

* Maven Surefire Plugin - The Surefire Plugin is used during the test phase of the build lifecycle to execute the unit tests of an application. We also need it to generate a report for our tests result
* Apache Maven Compiler Plugin - The Compiler Plugin is used to compile the sources of your project.
* selenium
* Testng

**Add plugins to project**Add the following code to project pom.xml(the file is located at the root of the project directory) file;

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

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.Inmar/2017/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>test\_id</groupId>

<artifactId>testId</artifactId>

<version>1.0-SNAPSHOT</version>

<!-- Add dependencies tag -->

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.testng/testng -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.2</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin -->

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.21.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin -->

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.7.0</version>

</dependency>

</dependencies>

</project>

Next, create a new package in <project\_name>/src/main/java/ and name it base(inside that package we'll create a class that would setup our test driver). When you're done, create a java class inside the base package and name it SetupTestDriver.

Add the following code to SetupTestDriver class

// src/main/java/base/SetupTestDriver.java

package base;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeOptions;

import org.openqa.selenium.firefox.FirefoxOptions;

import org.openqa.selenium.ie.InternetExplorerOptions;

import org.openqa.selenium.remote.RemoteWebDriver;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.concurrent.TimeUnit;

public class SetupTestDriver {

private WebDriver driver = null;

private String browser = null;

private String baseUrl = null;

private String os = null;

private String node = null;

public SetupTestDriver(String os, String browser, String baseUrl, String node) throws MalformedURLException {

this.browser = browser;

this.os = os;

this.baseUrl = baseUrl;

this.node = node;

Platform platform = Platform.fromString(os.toUpperCase());

if(browser.equalsIgnoreCase("chrome")) {

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.setCapability("platform", platform);

this.driver = new RemoteWebDriver(new URL(node + "/wd/hub"), chromeOptions);

} else if (browser.equalsIgnoreCase("firefox")) {

FirefoxOptions firefoxOptions = new FirefoxOptions();

firefoxOptions.setCapability("platform", platform);

this.driver = new RemoteWebDriver(new URL(node + "/wd/hub"), firefoxOptions);

} else {

InternetExplorerOptions ieOption = new InternetExplorerOptions();

ieOption.setCapability("platform", platform);

this.driver = new RemoteWebDriver(new URL(node + "/wd/hub"), ieOption);

}

this.driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);

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

this.driver.get(baseUrl);

}

public String getOs() {

return this.os;

}

public String getBrowser() {

return this.browser;

}

public String getBaseUrl() {

return this.baseUrl;

}

public String getNode() {

return this.node;

}

public WebDriver getDriver() {

return this.driver;

}

}

Now that we have SetupTestDriver clas all set up, let's create a simple java test to test Inmar search box.

Create a new test class in src/test/java, we can name it Inmartest

Add the following code to the class

// src/test/java/InmarSearchTest.java

import base.SetupTestDriver;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.Assert;

import org.testng.annotations.\*;

import java.net.MalformedURLException;

public class InmarSearchTest {

public WebDriver driver;

@BeforeClass(alwaysRun = true)

@Parameters({"os", "browser", "url", "node"})

public void setUp(String os, String browser, String url, String node) throws MalformedURLException {

SetupTestDriver setupTestDriver = new SetupTestDriver(os, browser, url, node);

driver = setupTestDriver.getDriver();

}

@Test

public void InmarTitleTest() {

// validate page title test

Assert.assertTrue(driver.getTitle().contentEquals("Inmar"));

}

@Test

public void InmarUrlTest() {

// validate current url test

Assert.assertTrue(driver.getCurrentUrl().contains("www.Inmar.com"));

}

@Test

public void InmarSearchButtonTest() {

// basic test to validate that search button is displayed, enabled and it's value

WebElement searchButtonElement = driver.findElement(By.name("btnK"));

Assert.assertTrue(searchButtonElement.isDisplayed());

Assert.assertTrue(searchButtonElement.isEnabled());

Assert.assertTrue(searchButtonElement.getAttribute("value").contains("InmarSearch"));

}

@Test

public void InmarFeelingLuckyButtonTest() {

// basic test to validate that feeling lucky button is displayed, enabled and it's value

WebElement feelingLuckyElement = driver.findElement(By.name("btnI"));

Assert.assertTrue(feelingLuckyElement.isDisplayed());

Assert.assertTrue(feelingLuckyElement.isEnabled());

Assert.assertTrue(feelingLuckyElement.getAttribute("value").contains("I'm Feeling Lucky"));

}

@Test

public void InmarSearchBox() {

// basic test to validate that search box displayed and enabled

WebElement searchElement = driver.findElement(By.name("q"));

Assert.assertTrue(searchElement.isDisplayed());

Assert.assertTrue(searchElement.isEnabled());

}

@AfterClass(alwaysRun = true)

public void closeBrowser() {

driver.quit();

}

}

Next, let set up our launch file - we'll use this file in running out test

Create a new resource package in src/test/, let's call it resources. Create a a new package inside resources, let's call it launchers- this will house our tests launchers files. Create another file Inmar.xml inside the launchers` package.

Add the following to Inmar.xml

<!--src/test/resources/launchers/Inmar.xml-->

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

<suite name="Inmar Search Tests" verbose="1" parallel="tests">

<parameter name="url" value="https://www.Inmar.com/"/>

<parameter name="os" value="windows"/>

<parameter name="node" value="<node\_url\_here>"/>

<test name="Tests on Firefox" preserve-order="true" group-by-instances="true">

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

<classes>

<class name="InmarSearchTest"/>

</classes>

</test>

<test name="Tests on IE11" preserve-order="true" group-by-instances="true">

<parameter name="browser" value="ie11"/>

<classes>

<class name="InmarSearchTest"/>

</classes>

</test>

<test name="Tests on Chrome" preserve-order="true" group-by-instances="true">

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

<classes>

<class name="InmarSearchTest"/>

</classes>

</test>

</suite>

Run

To run the test, open Inmar.xml and right click on it, then select Run