

## Assisted Practice: 2.3 Perform All Test Annotations

This section will guide you to:

- Implement @Test and other related Annotations

### Development Environment

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- JRE: OpenJDK Runtime Environment 11.0.2
- TestNG

This lab has six subsections, namely:

2.1.1 Creating a simple Java project

2.1.2 Installing TestNG

2.1.3 Adding TestNG libraries to the Class Path

2.1.4 Creating a class file named TestAnnotations

2.1.5 Running the project as TestNG

2.1.6 Pushing the code to your GitHub repositories

#### **Step 2.1.1:** Creating a simple Java project

- Open Eclipse
- Go the **File** menu. Choose **New->Java Project**
- Enter the project name as **Annotations**. Click on **Next**
- This will create the project files in the Project Explorer

### Step 2.1.2: Installing TestNG

- TestNG is installed as an eclipse plugin in your practice lab. (Refer FSD: Lab Guide - Phase 5)

### Step 2.1.3: Adding TestNG libraries to the Class Path

- In the Project Explorer, right click on **Annotations**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**
- Click on **Add Library**. Select **TestNG**. Click on **Next**. Click on **Finish**
- Click on **Apply and Close**

### Step 2.1.4: Creating a class file named TestAnnotations

- In the Project Explorer, expand **Annotations->Java Resources**
- Right click on **src** and choose **New->Class**
- In **Class Name**, enter **TestAnnotations**. In **Package Name**, enter **com.testannotations** and click on **Finish**
- Enter the following code:

```
package com.testannotations;
```

```
import org.testng.annotations.*;
```

```
public class TestAnnotations {
```

```
    @Test
```

```
    public void Test1() {
```

```

        System.out.println("Test1 Executed");
    }
    @Test
    public void Test2() {
        System.out.println("Test2 Executed");
    }

    @BeforeTest
    public void beforeTest() {
        System.out.println("BeforeTest Executed");
    }
    @AfterTest
    public void AfterTest() {
        System.out.println("AfterTest Executed");
    }

    @BeforeMethod
    public void beforeMethod() {
        System.out.println("BeforeMethod Executed");
    }
    @AfterMethod
    public void afterMethod() {
        System.out.println("AfterMethod Executed");
    }

    @BeforeClass
    public void beforeClass() {
        System.out.println("BeforeClass Executed");
    }
    @AfterClass
    public void afterClass() {
        System.out.println("AfterClass Executed");
    }
}

```

#### Step 2.1.5: Running the project as TestNG

- Right click on **TestAnnotations** class. Click on **TestNG->Convert to TestNG**
- Click on **Finish**. It will create a **TestNG.xml** file. Open that file
- Right click. Select **Run As ->TestNG Suite**

#### **Step 2.1.6:** Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**

## **Assisted Practice: 2.5 Group Test Cases and Parallel Test Execution**

This section will guide you to:

- Work with groups attribute of @Test
- Perform cross browser execution (parallel execution)

### **Development Environment**

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- JRE: OpenJDK Runtime Environment 11.0.2
- TestNG
- Selenium WebDriver Jar

This lab has eight subsections, namely:

2.2.1 Creating a simple Java project

2.2.2 Downloading Selenium WebDriver jar, chromedriver.exe, and geckodriver.exe

2.2.3 Adding the Web Driver dependency in the project

2.2.4 Installing TestNG

2.2.5 Adding TestNG libraries to the Class Path

2.2.6 Creating a Java class named ParallelTest.java

2.2.7 Running the project

2.2.8 Pushing the code to your GitHub repositories

### **Step 2.2.1: Creating a simple Java project**

- Open Eclipse
- Go the **File** menu. Choose **New->Java Project**

- Enter the project name as **Parallel Tests**. Click on **Next**
- This will create the project files in the Project Explorer

**Step 2.2.2:** Downloading Selenium WebDriver jar, chromedriver.exe, and geckodriver.exe

- Go to <https://www.seleniumhq.org/download/> to download the Selenium WebDriver dependency
- Under the section **Selenium Client & WebDriver Language Bindings**, click on **Download** for **Java client version: 3.141.59**
- On the same page, under **Third Party Drivers, Bindings, and Plugins**, click on **Latest** for **Mozilla Gecko Driver**
- Select the file suitable for your operating system
- Go back to the previous page. Click on **Latest** for **Google Chrome Driver**
- From the current releases, select the appropriate file per your Chrome version

**Step 2.2.3:** Adding the Web Driver dependency in the project

- In the Project Explorer, right click on **Parallel Tests**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**.
- Click on **Add External JARs** and browse the location where you have downloaded the JAR files
- Select JARs from the **root** folder and the **libs** folder
- Click on **Apply and Close**
- Copy the **chromedriver.exe** and **geckodriver.exe**, and paste it your project creating a resource folder

#### Step 2.2.4: Installing TestNG

- TestNG is installed as an eclipse plugin in your practice lab. (Refer FSD: Lab Guide - Phase 5)

#### Step 2.2.5: Adding TestNG libraries to the Class Path

- In the Project Explorer, right click on **Parallel Tests**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**
- Click on **Add Library**. Select **TestNG**. Click on **Next**. Click on **Finish**
- Click on **Apply and Close**

#### Step 2.2.6: Creating a Java class named ParallelTest.java

- In the Project Explorer, expand **Parallel Tests->Java Resources**
- Right click on **src** and choose **New->Class**
- In **Class Name**, enter **ParallelTests** and click on **Finish**. In **Package Name**, enter **com.parallel** and click on **Finish**
- Enter the following code:

```
package com.parallel;  
  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.openqa.selenium.firefox.FirefoxDriver;
```

```
import org.testng.annotations.Test;

public class ParallelTests {

    WebDriver driver;

    @Test(groups="Chrome")

    public void LaunchChrome() {

        System.setProperty("webdriver.chrome.driver",
"./Resources/chromedriver.exe");

        driver = new ChromeDriver();

        driver.get("https://www.facebook.com");

        try {

            Thread.sleep(2000);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    @Test(groups="Chrome", dependsOnMethods="LaunchChrome")

    public void TryFacebook1() {

        System.out.println(Thread.currentThread().getId());

        driver.findElement(By.id("email")).sendKeys("ravi10thstudent@gmail.com");

        driver.findElement(By.id("pass")).sendKeys("12345");

    }

}
```



```
driver.findElement(By.id("loginbutton")).click();

}

@Test(groups="Firefox")

public void LaunchFirefox() {

    System.setProperty("webdriver.gecko.driver", "./Resources/geckodriver.exe");

    driver = new FirefoxDriver();

    driver.get("https://www.facebook.com");

    try {

        Thread.sleep(4000);

    } catch (Exception e) {

        e.printStackTrace();

    }

}

@Test(groups="Firefox", dependsOnMethods="LaunchFirefox")

public void TryFacebook2() {

    System.out.println(Thread.currentThread().getId());

    driver.findElement(By.id("email")).sendKeys("ravi10thstudent@gmail.com");

    driver.findElement(By.id("pass")).sendKeys("ravi28394");
```

```
driver.findElement(By.id("loginbutton")).click();

System.out.println(Thread.currentThread().getId());

}

}
```

### Step 2.2.7 Running the project

- Right click on **ParallelTests** class. Click on **TestNG->Convert to TestNG**
- Click on **Finish**. It will create a **TestNG.xml** file. Open that file
- Right click. Select **Run As ->TestNG Suite**

### Step 2.2.8: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**

## Assisted Practice: 2.7 Evaluating Test Cases

This section will guide you to:

- Implement Soft and Hard Assertions on your test cases

### Development Environment

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- JRE: OpenJDK Runtime Environment 11.0.2
- Selenium WebDriver Jar
- TestNG

This lab has eight subsections, namely:

2.3.1 Creating a simple Java project

2.3.2 Downloading Selenium WebDriver jar, chromedriver.exe, and geckodriver.exe

2.3.3 Adding the WebDriver dependency in the project

2.3.4 Installing TestNG

2.3.5 Adding TestNG libraries to the Class Path

2.3.6 Creating a Java class named Assertions.java

2.3.7 Running the project

2.3.8 Pushing the code to your GitHub repositories

#### Step 2.3.1: Creating a simple Java project

- Open Eclipse
- Go the **File** menu. Choose **New->Java Project**
- Enter the project name as **Test Assertions**. Click on **Next**
- This will create the project files in the Project Explorer

**Step 2.3.2:** Downloading Selenium WebDriver jar, chromedriver.exe, and geckodriver.exe

- Go to <https://www.seleniumhq.org/download/> to download the **Selenium WebDriver** dependency
- Under the section **Selenium Client & WebDriver Language Bindings**, click on **Download** for **Java client version: 3.141.59**
- On the same page, under **Third Party Drivers, Bindings, and Plugins**, click on **Latest** for **Mozilla Gecko Driver**
- Select the file suitable for your operating system
- Go back to the previous page. Click on **Latest** for **Google Chrome Driver**
- From the current releases, select the appropriate file per your Chrome version

**Step 2.3.3:** Adding the WebDriver dependency in the project

- In the Project Explorer, right click on **Test Assertions**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**
- Click on **Add External JARs** and browse the location where you have downloaded the JAR files
- Select JARs from the **root** folder and the **libs** folder
- Click on **Apply and Close**
- Copy the chromedriver.exe and geckodriver.exe, and paste it your project creating a resource folder

**Step 2.3.4:** Installing TestNG

- TestNG is installed as an eclipse plugin in your practice lab. (Refer FSD: Lab Guide - Phase 5)

#### Step 2.3.5: Adding TestNG libraries to the Class Path

- In the Project Explorer, right click on **Test Assertions**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**
- Click on **Add Library**. Select **TestNG**. Click on **Next**. Click on **Finish**
- Click on **Apply and Close**

#### Step 2.3.6: Creating a Java class named ParallelTest.java

- In the Project Explorer, expand **Test Assertions->Java Resources**
- Right click on **src** and choose **New->Class**
- In **Class Name**, enter **Assertions** and click on **Finish**. In **Package Name**, enter **com.assert** and click on **Finish**
- Enter the following code:

```
package com.asserts;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;
```

```
public class Assertions {

    SoftAssert soft = new SoftAssert();

    WebDriver driver;

    @Test

    public void Launch() {

        System.setProperty("webdriver.chrome.driver",
"./Resources/chromedriver.exe");

        driver = new ChromeDriver();

        try {

            Thread.sleep(3000);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    @Test(dependsOnMethods = { "Launch" })

    public void Facebook() {

        driver.get("https://www.facebook.com");

        soft.assertEquals("FB Title", driver.getTitle());

        try {
```

```
        Thread.sleep(2000);

    } catch (Exception e) {

        e.printStackTrace();

    }

}

@Test(dependsOnMethods = { "Facebook" })

public void Login() {

    driver.findElement(By.id("email")).sendKeys("ravi10thstudent@gmail.com");

    driver.findElement(By.id("pass")).sendKeys("12345");

    driver.findElement(By.id("loginbutton")).click();

    soft.assertAll();

    try {

        Thread.sleep(3000);

    } catch (Exception e) {

        e.printStackTrace();

    }

}

}
```



### Step 2.3.7: Running the project

- Right click on **Assertions** class. Click on **TestNG->Convert to TestNG**
- Click on **Finish**. It will create a **TestNG.xml** file. Open that file
- Right click. Select **Run As ->TestNG Suite**

### Step 2.3.8: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**

## **Assisted Practice: 2.10 Integrating Selenium with Jenkins**

This section will guide you to:

- Integrate Selenium with Jenkins
- Configure Maven build

### **Development Environment**

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- JRE: OpenJDK Runtime Environment 11.0.2

This lab has ten subsections, namely:

- 2.4.1 Creating a Maven project
- 2.4.2 Editing the pom.xml and adding Selenium and JUnit dependencies
- 2.4.3 Creating a Java class named NewTest
- 2.4.4 Adding TestNG libraries to the Class Path
- 2.4.5 Converting the project into TestNG and changing the run configuration
- 2.4.6 Running the project as Maven test
- 2.4.7 Installing Jenkins
- 2.4.8 Adding Maven plugins to Jenkins
- 2.4.9 Adding the location of pom.xml in Jenkins CI Job
- 2.4.10 Pushing the code to your GitHub repositories

### **Step 2.4.1: Creating a Maven project**

- Open Eclipse

- Go the **File** menu. Choose **New->Other->Maven->Maven Project**
- On the **New Maven Project** dialog, select **Create a simple project** and click **Next**
- Enter **SeleJenk** in **Group Id** and **Artifact Id** and click on **Finish**

#### Step 2.4.2: Editing the pom.xml and adding Selenium and JUnit dependencies

- In the Project Explorer, expand the project **SeleJenk**
- Select **pom.xml** from **Project Explorer**
- Enter the following code:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/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>SeleJenk</groupId>

    <artifactId>SeleJenk</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <dependencies>

        <dependency>

            <groupId>junit</groupId>

            <artifactId>junit</artifactId>

            <version>3.8.1</version>

            <scope>test</scope>

        </dependency>

        <dependency>

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

            <artifactId>selenium-java</artifactId>

            <version>2.45.0</version>
```

```
</dependency>

<dependency>

    <groupId>org.testng</groupId>

    <artifactId>testng</artifactId>

    <version>6.14.2</version>

    <scope>test</scope>

</dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

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

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

            <version>3.6.0</version>

            <configuration>

                <goalPrefix>plugin</goalPrefix>

                <outputDirectory>target/dir</outputDirectory>

            </configuration>

        </plugin>

    </plugins>

</build>

</project>
```

### Step 2.4.3: Adding TestNG libraries to the Class Path

- In the Project Explorer, right click on **Test Assertions**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**
- Click on **Add Library**. Select **TestNG** (Refer FSD: Lab Guide - Phase 5). Click on **Next**. Click on **Finish**
- Click on **Apply and Close**

#### Step 2.4.4: Creating a TestNG class named NewTest

- In the Project Explorer, expand **Selejenk**
- Right click on **Selejenk**. Click on **New->Other->TestNG->TestNG Class**
- Enter **Package name** as **com.example** and **NewTest** in the **Name** textbox and click on **Finish**
- Enter the following code:

```
package com.example;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;

public class NewTest {
    private WebDriver driver;
    SoftAssert soft=new SoftAssert();

    @Test
    public void testEasy() {
        System.setProperty("webdriver.chrome.driver",
"./Resources/chromedriver.exe");
        driver=new ChromeDriver();
        driver.get("https://www.facebook.com");
        String title = driver.getTitle();
        soft.assertEquals("FB Login",title);
    }

    @BeforeTest
```

```
public void beforeTest() {  
    driver = new FirefoxDriver();  
}  
@AfterTest  
public void afterTest() {  
    driver.quit();  
}  
}
```

**Step 2.4.5:** Converting the project into TestNG and changing the run configuration

- In the Project Explorer, expand **SeleJenk**
- Right click on **SeleJenk** and choose **TestNG->convert to TestNG**

**Step 2.4.6:** Running the project as Maven test

- Right click on **SeleJenk**
- Click on **Run AS->Maven Test**

**Step 2.4.7:** Installing Jenkins

- Jenkins is already installed in your Practice lab.(Refer FSD: Lab Guide - Phase 5)
- Use the following commands to navigate to the above-mentioned directory.

```
cd /usr/share  
ls
```

**Step 2.4.8:** Adding Maven plugins to Jenkins

- In the Jenkins dashboard, click on **Manage Jenkins**
- Click on **Manage Plugins**

- Select the **Available** tab, then find the **Maven Integration** plugin
- Click **Install** without restart

#### **Step 2.4.9:** Adding the location of pom.xml in Jenkins CI Job

- Click on **New Item** to create **CI Job**
- Select the **Maven project radio** button and enter **Item Name** as **SeleJenk**
- Click on **Build Environment**
- In **Root POM**, specify the location of pom.xml from your Eclipse workspace
- In **Goals and Options**, type **clean test**. Click on **Save**
- Click on the **SeleJenk** project page and click on the **Build Now** link

#### **Step 2.4.10:** Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**



# Assisted Practice: 2.12 Set Up Selenium Grid

This section will guide you to:

- Set up Selenium Grid, which includes setup of Hub and Nodes

This lab has three subsections, namely:

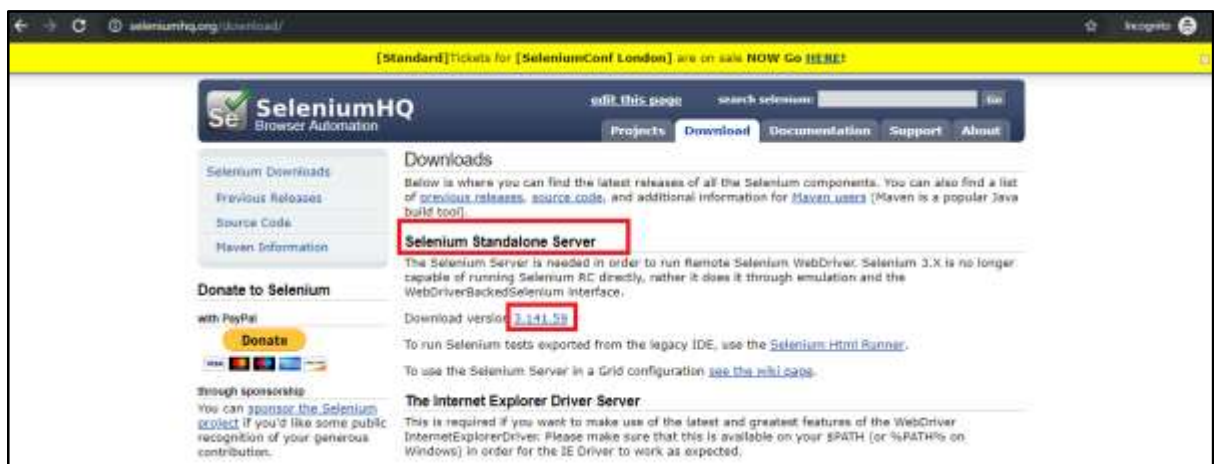
2.5.1 Setting up Selenium Grid hub

2.5.2 Setting up Selenium Grid Nodes

2.5.3 Pushing the code to your GitHub repositories

## **Step 2.5.1:** Setting up Selenium Grid hub

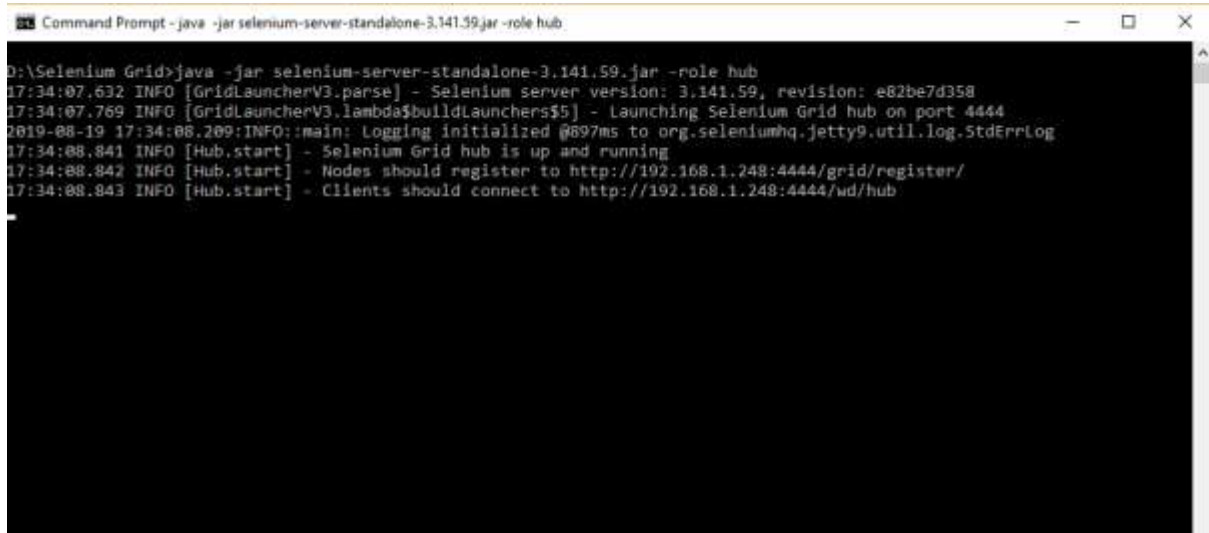
- a. Download Selenium standalone Server jar file from <https://www.seleniumhq.org/download/> link



- b. Save it in a folder
- c. Go to command prompt
- d. Navigate to folder structure where you have saved the Selenium standalone Server jar file

e. Type the below command in command prompt

**Java -jar selenium-server-standalone-3.141.59.jar -role hub** and click on **Enter** button



```
Command Prompt - java -jar selenium-server-standalone-3.141.59.jar -role hub
D:\Selenium Grid>java -jar selenium-server-standalone-3.141.59.jar -role hub
17:34:07.632 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
17:34:07.769 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444
2019-08-19 17:34:08.209:INFO::main: Logging initialized @697ms to org.seleniumhq.jetty9.util.log.StdErrLog
17:34:08.841 INFO [Hub.start] - Selenium Grid hub is up and running
17:34:08.842 INFO [Hub.start] - Nodes should register to http://192.168.1.248:4444/grid/register/
17:34:08.843 INFO [Hub.start] - Clients should connect to http://192.168.1.248:4444/wd/hub
```

f. Open the Chrome browser

g. Enter URL as **http://localhost:4444/grid/console** and click on **Enter**

h. Grid console page is loaded as below



### Step 2.5.2: Setting up the Selenium Grid Nodes

a. Once the Selenium Grid Hub is set up, the next step is to set up Selenium Grid nodes.

- b. Open the new command prompt
- c. Navigate to the folder structure where you have saved the Selenium standalone server jar file
- d. Type the below command in command prompt

**java -jar selenium-server-standalone-3.141.59.jar -role node -hub**

**http://localhost:4444/grid/register** and click on **Enter** button, which looks like

```
Command Prompt - java -jar selenium-server-standalone-3.141.59.jar -role node -hub http://localhost:4444/grid/register
Microsoft Windows [Version 10.0.17134.950]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Selenium Grid

D:\Selenium Grid>java -jar selenium-server-standalone-3.141.59.jar -role node -hub http://localhost:4444/grid/register
17:54:25.329 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
17:54:25.499 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 28717
2019-08-19 17:54:26.247:INFO::main: logging initialized @1328ms to org.seleniumhq.jetty9.util.log.StdErrLog
17:54:26.566 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
17:54:26.655 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 28717
17:54:26.656 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub
17:54:26.939 INFO [SelfRegisteringRemote$1.run] - Starting auto registration thread. Will try to register every 5000 ms.
17:54:30.868 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/register
17:54:31.159 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
```

- e. Open the browser
- f. Enter URL as **http://localhost:4444/grid/console** and click on **Enter**
- g. Grid console page is loaded below, which shows **Browsers** by default



- h. Click on **Configuration**, which shows Configuration details



### Step 2.5.3: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**

## Assisted Practice: 2.13 Grid Configuration Using JSON

This section will guide you to:

- Configure the grid using JSON

This lab has mainly three subsections, namely:

- 2.6.1 Configuring the grid hub using JSON
- 2.6.2 Configuring the grid nodes using JSON
- 2.6.3 Pushing the code to your GitHub repositories

### **Step 2.6.1:** Configuring the grid hub using JSON

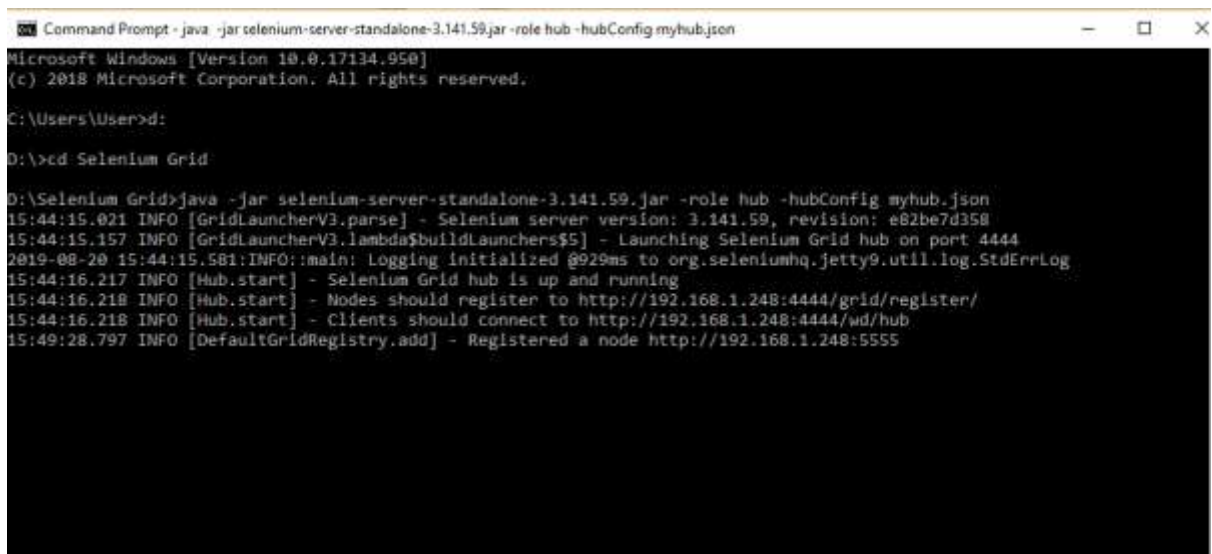
- a. Create JSON file for the hub which looks like

```
{
  "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
}
```

- b. Save it in a folder with a valid name (example: myhub) in which we have saved Selenium standalone Server jar file
- c. Go to command prompt

- d. Navigate to folder structure where you have saved the Selenium standalone Server jar file
- e. Type the below command in command prompt

**Java -jar selenium-server-standalone-3.141.59.jar -role hub -hubConfig myhub.json** and click on **Enter** button, which looks like



```
Command Prompt - java -jar selenium-server-standalone-3.141.59.jar -role hub -hubConfig myhub.json
Microsoft Windows [Version 10.0.17134.950]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Selenium Grid

D:\Selenium Grid>java -jar selenium-server-standalone-3.141.59.jar -role hub -hubConfig myhub.json
15:44:15.021 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d350
15:44:15.157 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444
2019-08-20 15:44:15.581:INFO::main: Logging initialized @929ms to org.seleniumhq.jetty9.util.log.StdErrLog
15:44:16.217 INFO [Hub.start] - Selenium Grid hub is up and running
15:44:16.218 INFO [Hub.start] - Nodes should register to http://192.168.1.248:4444/grid/register/
15:49:28.797 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.1.248:5555
```

- f. Open the Chrome browser
- g. Enter URL as 'http://localhost:4444/grid/console' and click on enter
- h. Grid console page is loaded as below



### Step 2.6.2: Configuring the grid nodes using JSON

- a. Once the Selenium Grid Hub using JSON is configured, the next step is to configure Selenium Grid nodes using JSON.
- b. Create JSON file for node, which looks like:

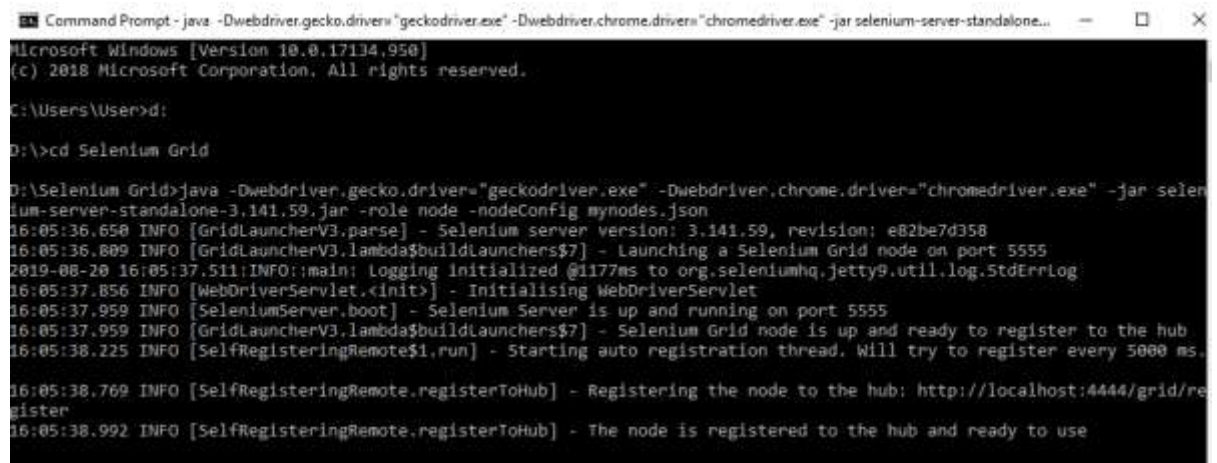
```
{
  "capabilities": [
    {
      "browserName": "firefox",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "chrome",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    }
  ],
  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
  "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": {}
}
```

- c. Save it in a folder with a valid name (example: mynode) in which we have saved Selenium standalone Server jar file
- d. Open the new command prompt
- e. Navigate to the folder structure where you have saved the Selenium standalone server jar file
- f. Type the below command in command prompt



**java -Dwebdriver.gecko.driver="geckodriver.exe" -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -nodeConfig mynodes.json**

and click on **Enter** button, which looks like



```
Command Prompt - java -Dwebdriver.gecko.driver="geckodriver.exe" -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -nodeConfig mynodes.json
Microsoft Windows [Version 10.0.17134.950]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Selenium Grid

D:\Selenium Grid>java -Dwebdriver.gecko.driver="geckodriver.exe" -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -nodeConfig mynodes.json
16:05:36.658 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
16:05:36.809 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 5555
2019-08-20 16:05:37.511:INFO::main: logging initialized @1177ms to org.seleniumhq.jetty9.util.log.StdErrLog
16:05:37.856 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
16:05:37.959 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 5555
16:05:37.959 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub
16:05:38.225 INFO [SelfRegisteringRemote.run] - Starting auto registration thread. Will try to register every 5000 ms.
16:05:38.769 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/register
16:05:38.992 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
```

- g. Open the browser
- h. Enter URL as **http://localhost:4444/grid/console** and click on **Enter**
- i. Grid console page is loaded, which shows **Browsers** by default



- j. Click on **Configuration** which shows Configuration details



### Step 2.6.3: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**

## Assisted Practice: 2.14 Running Tests on Selenium Grid

This section will guide you to:

- Run the scripts on Selenium grid

This lab has two subsections, namely:

2.7.1 Running the tests on Selenium grid

2.7.2 Pushing the code to your GitHub repositories

### **Step 2.7.1:** Running the Tests on Selenium grid

- Open Eclipse
- Click on **Package** and navigate to **New --> Class**
- Give a valid Class name (example: GridTest)
- Check the **public static void main** checkbox and click on **finish**, which will create a blank Java class
- Write the desired capabilities in the class, which looks like

```

package testing.sidTesting;

import org.openqa.selenium.Platform;
import org.openqa.selenium.remote.DesiredCapabilities;

public class GridTest {

    public static void main(String[] args) {
        DesiredCapabilities cap = new DesiredCapabilities();
        cap.setBrowserName("chrome");
        cap.setPlatform(Platform.WIN10);
    }
}

```

- Start the selenium grid hub in command prompt using **java -jar selenium-server-standalone-3.141.59.jar -role hub** command
- Start the selenium grid node in Command prompt using **java -Dwebdriver.chrome.driver="chromedriver.exe -jar selenium-server-standalone-3.141.59.jar -role node -hub http://localhost:4444/grid/register** command
- Go to eclipse and add a statement for remoteWebdriver, which has an implementation of WebDriver, should pass the hub port (http://192.168.1.248:4444/wd/hub), and DesiredCapabilities object as parameters
- Write Selenium code to open the browser and navigate to any web page (example: Google page)

```

import java.net.URL;

import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class GridTest {

    public static void main(String[] args) throws MalformedURLException {
        DesiredCapabilities cap = new DesiredCapabilities();
        cap.setBrowserName("chrome");
        cap.setPlatform(Platform.WIN10);

        URL url = new URL("http://192.168.1.248:4444/wd/hub");
        WebDriver driver = new RemoteWebDriver(url, cap);

        driver.get("https://www.google.com");
        System.out.println("Google Title: " + driver.getTitle());

        driver.close();
    }
}

```

- Execute the Java program by right-clicking on the program and navigating to **Run As--> 1 Java Application**
- This is how it looks like in Eclipse console

```

Aug 21, 2019 5:14:13 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Google Title: Google

```

- We can see the capabilities passed through are displayed in both command prompts in server (hub) as well as in clients (node)
- Selenium grid hub in command prompt with desired capabilities looks like

```
C:\Windows\System32\cmd.exe - java -jar selenium-server-standalone-3.141.59.jar -role hub
Microsoft Windows [Version 10.0.17134.958]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Selenium Grid>java -jar selenium-server-standalone-3.141.59.jar -role hub
17:01:00.889 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
17:01:01.317 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444
2019-08-21 17:01:02.004 INFO: main: logging initialized @1822ms to org.seleniumhq.jetty9.util.log.Slf4jring
17:01:03.315 INFO [Hub.start] - Selenium Grid hub is up and running
17:01:03.317 INFO [Hub.start] - Nodes should register to http://192.168.1.148:4444/grid/register/
17:01:03.317 INFO [Hub.start] - Clients should connect to http://192.168.1.148:4444/wd/hub
17:01:13.072 INFO [DefaultGridRegistry.app] - Registered a node http://192.168.1.148:35482
17:14:02.072 INFO [RequestHandler.process] - Got a request to create a new session; Capabilities {browserName: chrome, platform: LINUX}
17:14:02.078 INFO [TestSlot.getNewSession] - Trying to create a new session on test slot (server:COMF16_UUID=64178522-5111-44a4-b432-a61ec8b4543a, seleniumProtocol=WebDriver, browserName:chrome, maxInstances:5, platformName:LINUX, platformVersion:)
```

- Selenium grid node in command prompt with desired capabilities looks like

```
C:\Windows\System32\cmd.exe - java -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -hub http://localhost:4444/grid/register
Microsoft Windows [Version 10.0.17134.958]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Selenium Grid>java -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -hub http://localhost:4444/grid/register
17:06:07.388 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
17:06:07.869 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching a Selenium Grid node on port 35482
2019-08-21 17:06:08.786 INFO: main: logging initialized @5468ms to org.seleniumhq.jetty9.util.log.Slf4jring
17:06:09.242 INFO [WebDriverServerV2.start] - Initializing WebDriverService
17:06:09.316 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 35482
17:06:09.376 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Selenium Grid node is up and ready to register to the hub
17:06:09.842 INFO [SelfRegisteringRemote.registerToHub] - Starting auto registration thread. Will try to register every 5000 ms.
17:06:10.776 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/register
17:06:11.071 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
17:14:02.722 INFO [ActiveSessionFactory.app] - Capabilities are: {
  "browserName": "chrome",
  "platform": "LINUX"
}
17:14:02.764 INFO [ActiveSessionFactory.lambda$apply$11] - Matched factory org.openqa.selenium.grid.session.remote.RemoteSessionFactory (provider: org.openqa.selenium.chrome.ChromeDriverService)
Starting ChromeDriver 78.0.3900.10 (620c94080514d51307672c15116d5-rwfs/branch-heads/3800@#004) on port 6406
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test frameworks to prevent access by malicious code.
17:14:12.354 INFO [ProtocolHandshake.createSession] - Detected dialect: W3C
17:14:12.073 INFO [RemoteSessionFactory.lambda$performHandshake$0] - Started new session 5ca027be5785c383a19527102f35bdab (org.openqa.selenium.chrome.ChromeDriverService)
```

## Step 2.7.2: Pushing the code to your GitHub repositories

- Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

- Initialize your repository using the following command:

**git init**

- Add all the files to your git repository using the following command:

**git add .**

- Commit the changes using the following command:

**git commit . -m "Changes have been committed."**

- Push the files to the folder you initially created using the following command:

**git push -u origin master**