Testing web apps in the cloud (Sauce Labs) using Selenium and TestNG in Java

Overview

In this tutorial, we will create some tests in TestNG in order to validate a simple browser interaction using Sauce Labs for cloud testing.

Please note

Within this tutorial, only one Test Execution will be used; it will contain one Test Run with all the results for the different used browsers. Thus, the overall test run status will be affected by the results made for all the browsers.

Instead of this approach, a different one could be creating a Test Execution per each browser; this would require some adaptions in order to produce a XML report per each used browser. This approach would give the ability to take advantage of Test Environments (more info in Working with Test Environments).

Requirements

- Install Java
- Install all dependencies using "mvn"

Description

This tutorial is based on Sauce Labs's own tutorial for TestNG.

You may start by cloning the repository https://github.com/saucelabs-sample-test-frameworks/Java-TestNG-Selenium.

```
git clone https://github.com/saucelabs-sample-test-frameworks/Java-TestNG-Selenium
```

There are two tests, which extend a TestBase helper class where the target browsers/capabilities are enumerated along with the logic for obtaining the Sauce Labs credentials.

src/test/java/com/yourcompany/Tests/TestBase.java

```
package com.yourcompany.Tests;

import com.saucelabs.common.SauceOnDemandAuthentication;
import com.saucelabs.common.SauceOnDemandSessionIdProvider;
import com.saucelabs.testng.SauceOnDemandAuthenticationProvider;
import com.saucelabs.testng.SauceOnDemandTestListener;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.ITestResult;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Listeners;

import java.lang.reflect.Method;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.UnexpectedException;

/**
 * Simple TestNG test which demonstrates being instantiated via a DataProvider in order to supply multiple
browser combinations.
```

```
 *
 * @author Neil Manvar
 */
public class TestBase  {

    public String buildTag = System.getenv("BUILD_TAG");

    public String username = System.getenv("SAUCE_USERNAME");

    public String accesskey = System.getenv("SAUCE_ACCESS_KEY");

    /**
     * ThreadLocal variable which contains the  {@link WebDriver} instance which is used to perform browser
interactions with.
     */
    private ThreadLocal<WebDriver> webDriver = new ThreadLocal<WebDriver>();

    /**
     * ThreadLocal variable which contains the Sauce Job Id.
     */
    private ThreadLocal<String> sessionId = new ThreadLocal<String>();

    /**
     * DataProvider that explicitly sets the browser combinations to be used.
     *
     * @param testMethod
     * @return Two dimensional array of objects with browser, version, and platform information
     */
    @DataProvider(name = "hardCodedBrowsers", parallel = true)
    public static Object[][] sauceBrowserDataProvider(Method testMethod) {
        return new Object[][]{
                new Object[]{"MicrosoftEdge", "14.14393", "Windows 10"},
                new Object[]{"firefox", "49.0", "Windows 10"},
                new Object[]{"internet explorer", "11.0", "Windows 7"},
                new Object[]{"safari", "10.0", "OS X 10.11"},
                new Object[]{"chrome", "54.0", "OS X 10.10"},
                new Object[]{"firefox", "latest-1", "Windows 7"},
        };
    }

    /**
     * @return the {@link WebDriver} for the current thread
     */
    public WebDriver getWebDriver() {
        return webDriver.get();
    }

    /**
     *
     * @return the Sauce Job id for the current thread
     */
    public String getSessionId() {
        return sessionId.get();
    }

    /**
     * Constructs a new {@link RemoteWebDriver} instance which is configured to use the capabilities defined by
the browser,
     * version and os parameters, and which is configured to run against ondemand.saucelabs.com, using
     * the username and access key populated by the {@link #authentication} instance.
     *
     * @param browser Represents the browser to be used as part of the test run.
     * @param version Represents the version of the browser to be used as part of the test run.
     * @param os Represents the operating system to be used as part of the test run.
     * @param methodName Represents the name of the test case that will be used to identify the test on Sauce.
     * @return
     * @throws MalformedURLException if an error occurs parsing the url
     */
    protected void createDriver(String browser, String version, String os, String methodName)
            throws MalformedURLException, UnexpectedException {
        DesiredCapabilities capabilities = new DesiredCapabilities();
```

```
        // set desired capabilities to launch appropriate browser on Sauce
        capabilities.setCapability(CapabilityType.BROWSER_NAME, browser);
        capabilities.setCapability(CapabilityType.VERSION, version);
        capabilities.setCapability(CapabilityType.PLATFORM, os);
        capabilities.setCapability("name", methodName);

        if (buildTag != null) {
            capabilities.setCapability("build", buildTag);
        }

        // Launch remote browser and set it as the current thread
        webDriver.set(new RemoteWebDriver(
                new URL("https://" + username + ":" + accesskey + "@ondemand.saucelabs.com:443/wd/hub"),
                capabilities));

        // set current sessionId
        String id = ((RemoteWebDriver) getWebDriver()).getSessionId().toString();
        sessionId.set(id);
    }

    /**
     * Method that gets invoked after test.
     * Dumps browser log and
     * Closes the browser
     */
    @AfterMethod
    public void tearDown(ITestResult result) throws Exception {
        ((JavascriptExecutor) webDriver.get()).executeScript("sauce:job-result=" + (result.isSuccess() ?
"passed" : "failed"));
        webDriver.get().quit();
    }

    protected void annotate(String text) {
        ((JavascriptExecutor) webDriver.get()).executeScript("sauce:context=" + text);
    }
}
```

The tests use the Page Objects pattern, implemented by the following class.

src/test/java/com/yourcompany/Pages/GuineaPigPage.java

```
package com.yourcompany.Pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class GuineaPigPage {

    @FindBy(linkText = "i am a link")
    private WebElement theActiveLink;

    @FindBy(id = "your_comments")
    private WebElement yourCommentsSpan;

    @FindBy(id = "comments")
    private WebElement commentsTextAreaInput;

    @FindBy(id = "submit")
    private WebElement submitButton;

    public WebDriver driver;
    public static String url = "https://saucelabs-sample-test-frameworks.github.io/training-test-page";

    public static GuineaPigPage visitPage(WebDriver driver) {
        GuineaPigPage page = new GuineaPigPage(driver);
        page.visitPage();
        return page;
    }

    public GuineaPigPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public void visitPage() {
        this.driver.get(url);
    }

    public void followLink() {
        theActiveLink.click();
    }

    public void submitComment(String text) {
        commentsTextAreaInput.sendKeys(text);
        submitButton.click();

        // Race condition for time to populate yourCommentsSpan
        WebDriverWait wait = new WebDriverWait(driver, 15);
        wait.until(ExpectedConditions.textToBePresentInElement(yourCommentsSpan, text));
    }

    public String getSubmittedCommentText() {
        return yourCommentsSpan.getText();
    }

    public boolean isOnPage() {
        String title = "I am a page title - Sauce Labs";
        return driver.getTitle() == title;
    }

}
```

Test 1: Verify the link works on the "GuineaPigPage"

src/test/java/com/yourcompany/Tests/FollowLinkTest.java

```
package com.yourcompany.Tests;

import com.yourcompany.Pages.GuineaPigPage;
import org.openqa.selenium.InvalidElementStateException;
import org.openqa.selenium.WebDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

import java.lang.reflect.Method;
import java.net.MalformedURLException;
import java.rmi.UnexpectedException;

/**
 * Created by mehmetgerceker on 12/7/15.
 */

public class FollowLinkTest extends TestBase {

    /**
     * Runs a simple test verifying link can be followed.
     *
     * @throws InvalidElementStateException
     */
    @Test(dataProvider = "hardCodedBrowsers")
    public void verifyLinkTest(String browser, String version, String os, Method method)
            throws MalformedURLException, InvalidElementStateException, UnexpectedException {

        //create webdriver session
        this.createDriver(browser, version, os, method.getName());
        WebDriver driver = this.getWebDriver();

        this.annotate("Visiting GuineaPig page...");
        GuineaPigPage page = GuineaPigPage.visitPage(driver);

        this.annotate("Clicking on link...");
        page.followLink();

        this.annotate("Asserting that we are on a new page...");
        Assert.assertFalse(page.isOnPage());
    }

}
```

Test 2: Verify comments can be added on the "GuineaPigPage"

src/test/java/com/yourcompany/Tests/TextInputTest.java

```
package com.yourcompany.Tests;

import com.yourcompany.Pages.GuineaPigPage;
import org.openqa.selenium.InvalidElementStateException;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.testng.Assert;

import java.lang.reflect.Method;
import java.net.MalformedURLException;
import java.rmi.UnexpectedException;
import java.util.UUID;


/**
 * Created by mehmetgerceker on 12/7/15.
 */

public class TextInputTest extends TestBase {

    /**
     * Runs a simple test verifying if the comment input is functional.
     * @throws InvalidElementStateException
     */
    @org.testng.annotations.Test(dataProvider = "hardCodedBrowsers")
    public void verifyCommentInputTest(String browser, String version, String os, Method method)
            throws MalformedURLException, InvalidElementStateException, UnexpectedException {
        this.createDriver(browser, version, os, method.getName());
        WebDriver driver = this.getWebDriver();

        String commentInputText = UUID.randomUUID().toString();

        this.annotate("Visiting GuineaPig page...");
        GuineaPigPage page = GuineaPigPage.visitPage(driver);

        this.annotate(String.format("Submitting comment: \"%s\"", commentInputText));
        page.submitComment(commentInputText);

        this.annotate(String.format("Asserting submitted comment is: \"%s\"", commentInputText));
        Assert.assertTrue(page.getSubmittedCommentText().contains(commentInputText));
    }

}
```

Please note

If you wish to map the test method to an existing Test issue, or to create links to requirements whenever importing the results, please have a look at the tutorial Testing using TestNG in Java which provides the necessary instructions in order to setup TestNG for this purpose.

Before running the tests themselves, you need to export some environment variables with your Sauce Lab's username along with the respective access key, which you can obtain from within the User Settings section in your Sauce Lab's profile page.

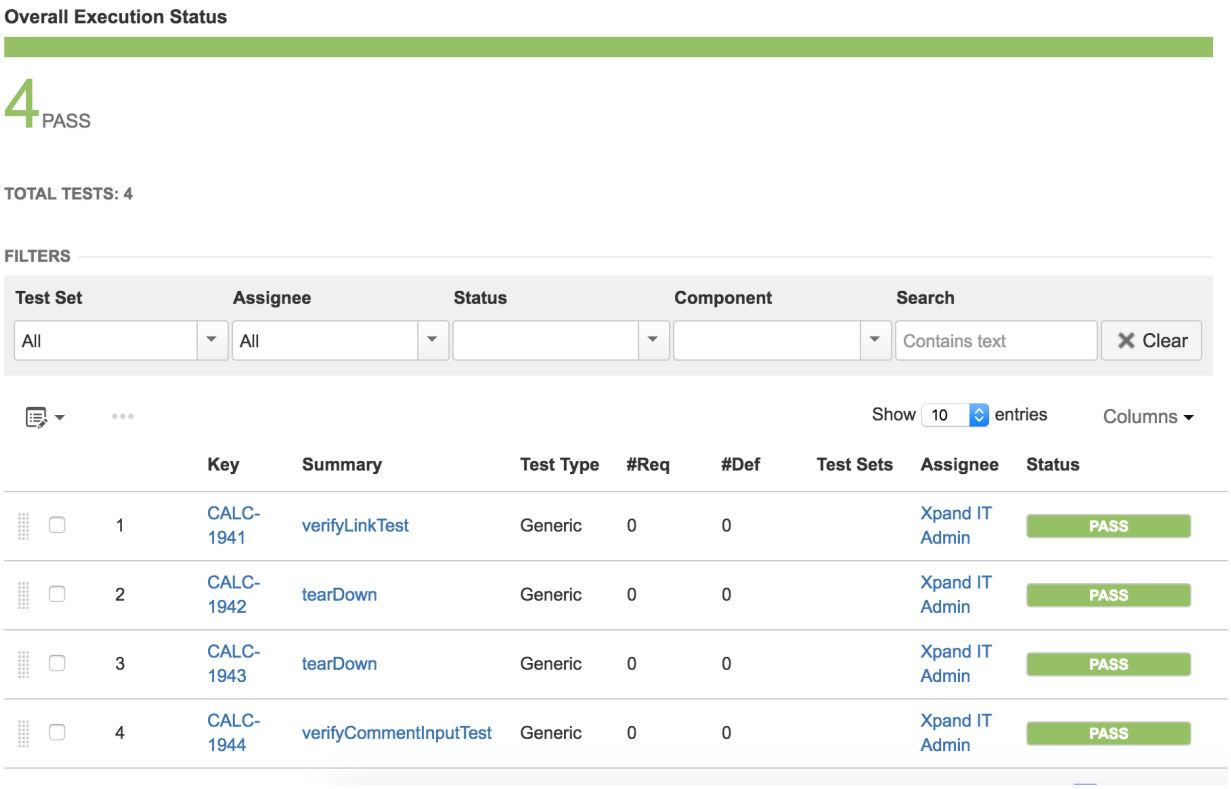
```
export SAUCE_USERNAME=<your Sauce Labs username>
export SAUCE_ACCESS_KEY=<your Sauce Labs access key>
```

Test(s) then can be run in parallel using Maven's "test" task.

```
mvn test
```

After successfully running the tests and generating the TestNG XML report (e.g. testng-results.xml), it can be imported to Xray (either by the REST API or through the **Import Execution Results** action within the Test Execution).

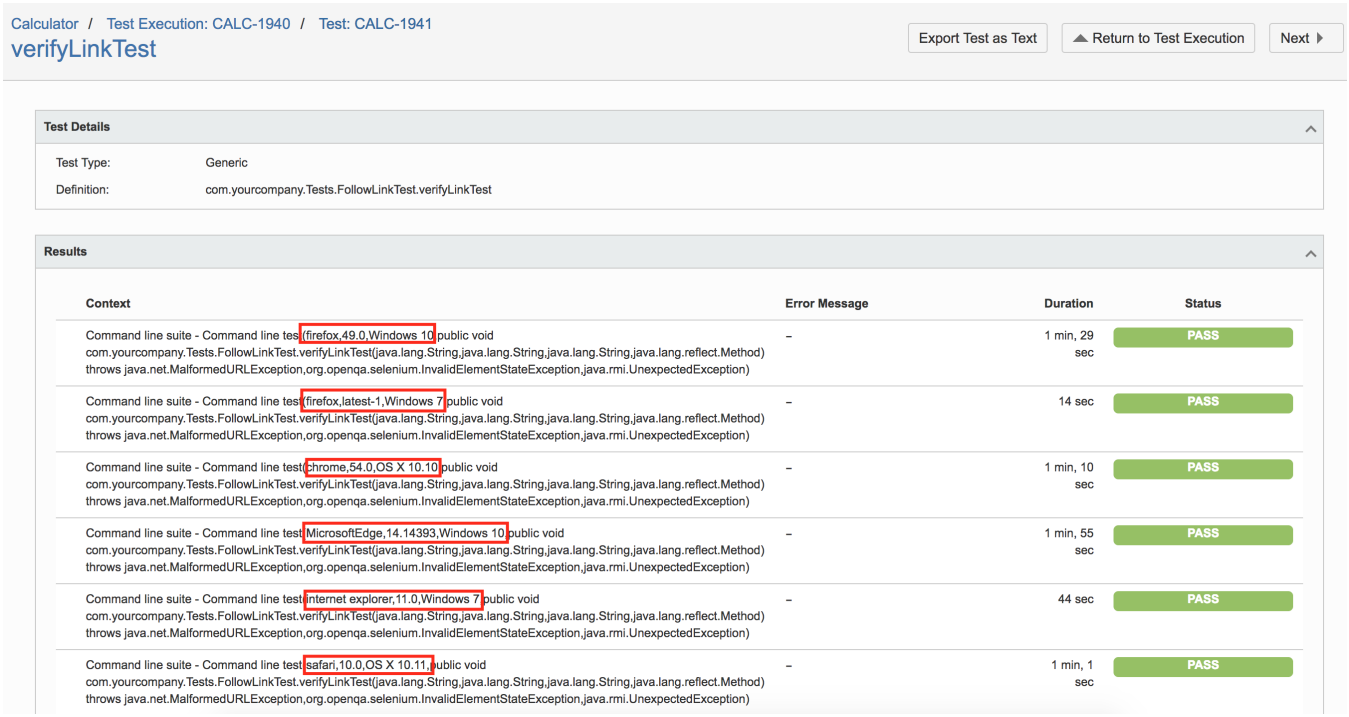
```
curl -H "Content-Type: multipart/form-data" -u user:password -F "file=@target/surefire-reports/testng-results.xml" "https://yourjiraserver/rest/raven/1.0/import/execution/testng?projectKey=CALC"
```



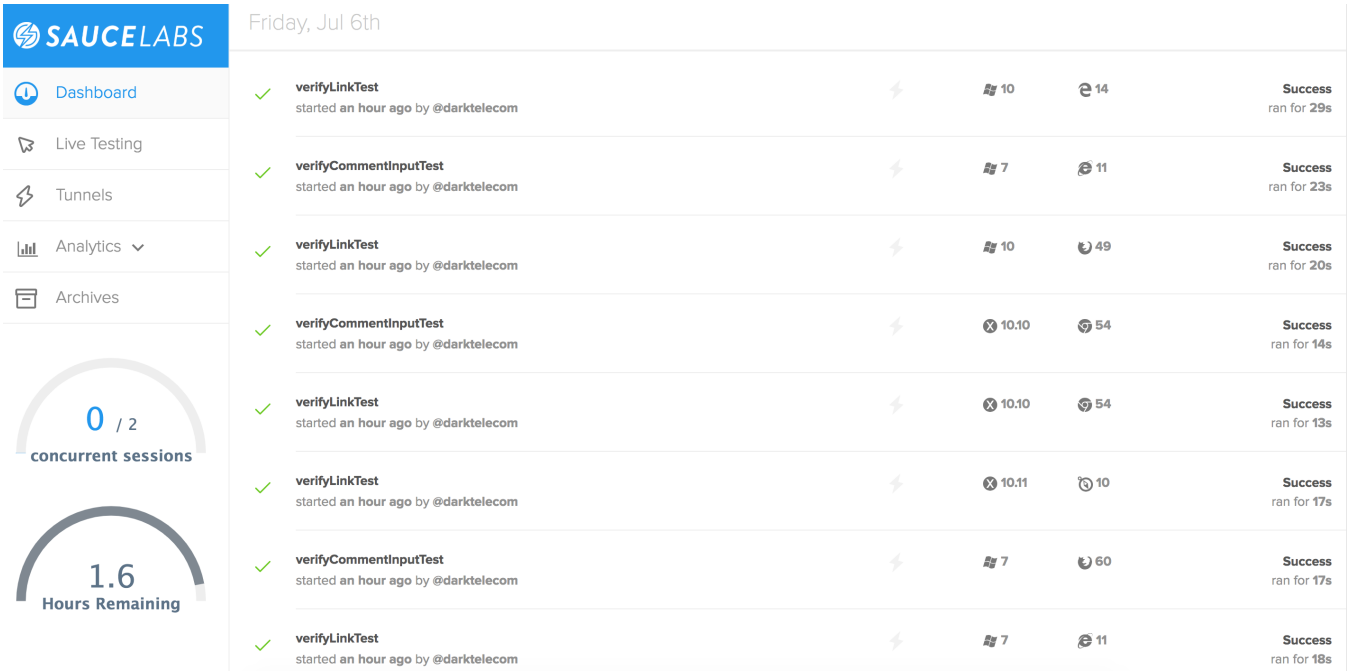
TestNG's tests are mapped to Generic Tests in Jira, and the **Generic Test Definition** field contains the namespace, the name of the class, and the method name that implements the Test case.

The execution screen details will not only provide information on the overall test run result, but also on a "browser" basis.

For each browser, a different "context" will appear along with the respective result.



In Sauce Labs you can see some info about it.



References

- https://github.com/saucelabs-sample-test-frameworks/Java-TestNG-Selenium
- https://wiki.saucelabs.com/display/DOCS/Instant+Selenium+Java+Tests