SELENIUM 101

Testing at the speed of awesome

Course Administration

- This session is ~3 hours long
  - 10 min. breaks every hour
- Slides, demonstrations and exercises
- PDF copy of the materials is available
- Ask questions at any time
  - Use the chat window

The Training Environment

- ReadyTech virtual machines
- Eclipse
- Maven

Saucelabs.com Account

If you haven't do so already, please take the time to create a saucelabs.com account.

If you had already made one in the past, and your free trial has run out, let me know!

Accessing the ReadyTech Environment

- Check your email for a link to the environment and your access code.
  - Look in spam!
- Enter your access code.
- You will arrive in the Lobby tab.
- Access the environment by clicking on the Lab tab.
- Click on the Remote Desktop image. This will take to you to the remote desktop.

Agenda

- Intro to Sauce Labs
- Selenium Test Script Basics
- Writing Selenium Scripts
- Selenium Scripts With Sauce Labs
- Introduction to Testing Frameworks
- Testing With Sauce Labs: Best Practices

INTRODUCTION TO SAUCE LABS AND SELENIUM

Module Objectives

This module enables you to:

- Understand how Sauce Labs fits into the CI/CD Life Cycle
- Understand what Selenium is and how it is used with Sauce Labs
- Use the Sauce Labs UI

What is Sauce Labs?

Testing infrastructure in the cloud for web and mobile web applications.

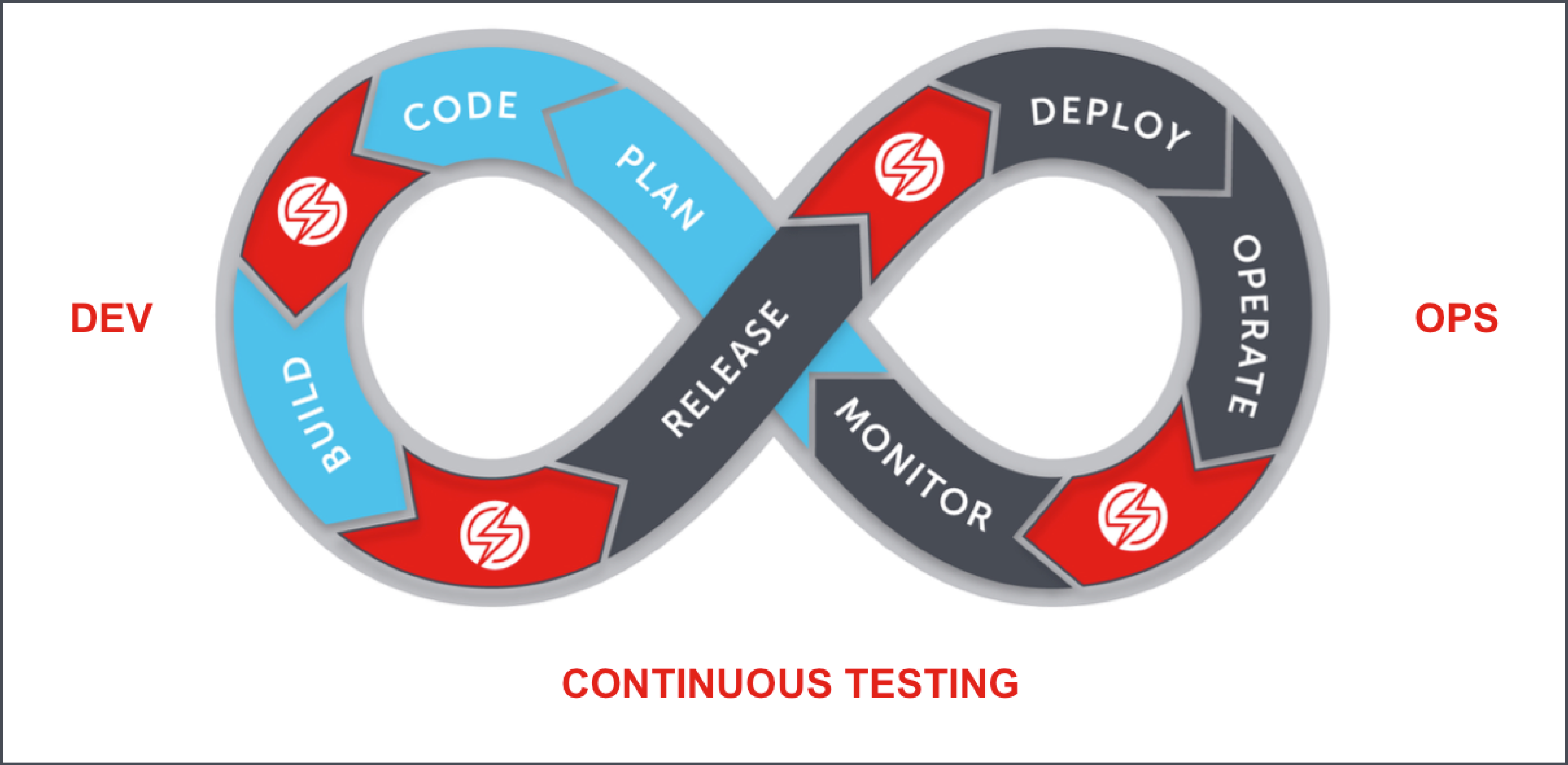
- Web Automated Testing
- Mobile Automated Testing

Sauce Labs History

Founded in 2008, by Steven Hazel, John Dunham, and Jason Huggins (Co-creator of Selenium).

```
"Our purpose is to revolutionize testing
      so that development teams
   are free to innovate and deliver
      amazing applications—faster".
```

CI/CD Cycle



Continuous Integration/Continuous Development

What is Selenium?

Developed in 2004 by Jason Huggins at ThoughtWorks.

Browser automation framework used for testing.

- Simulate site interaction
- Automate Application Testing

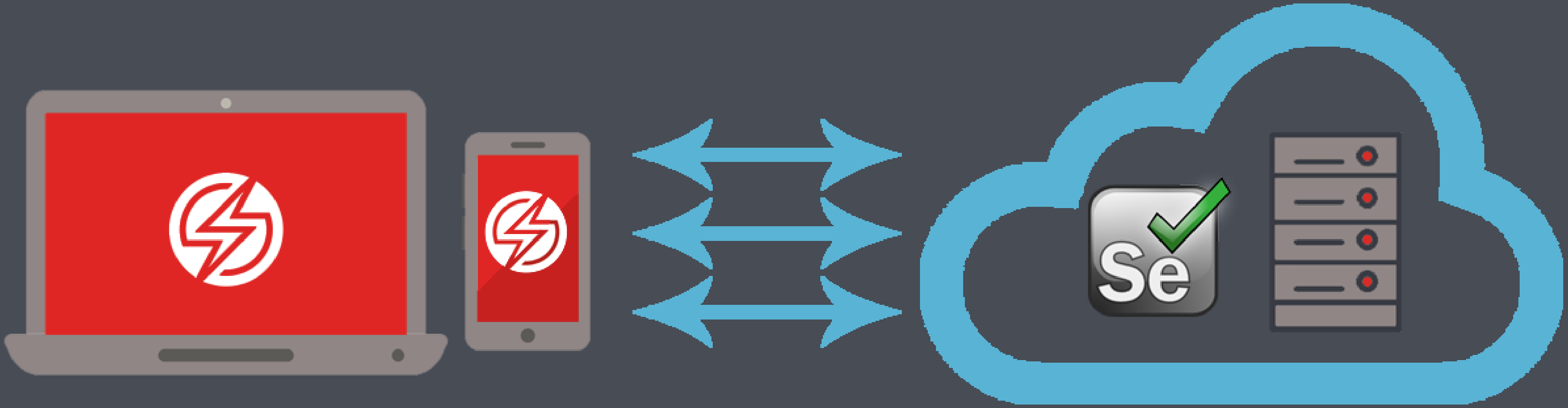
Local vs Remote Selenium Tests

Local

- Test on local OS and browsers only
- Record results

Remote

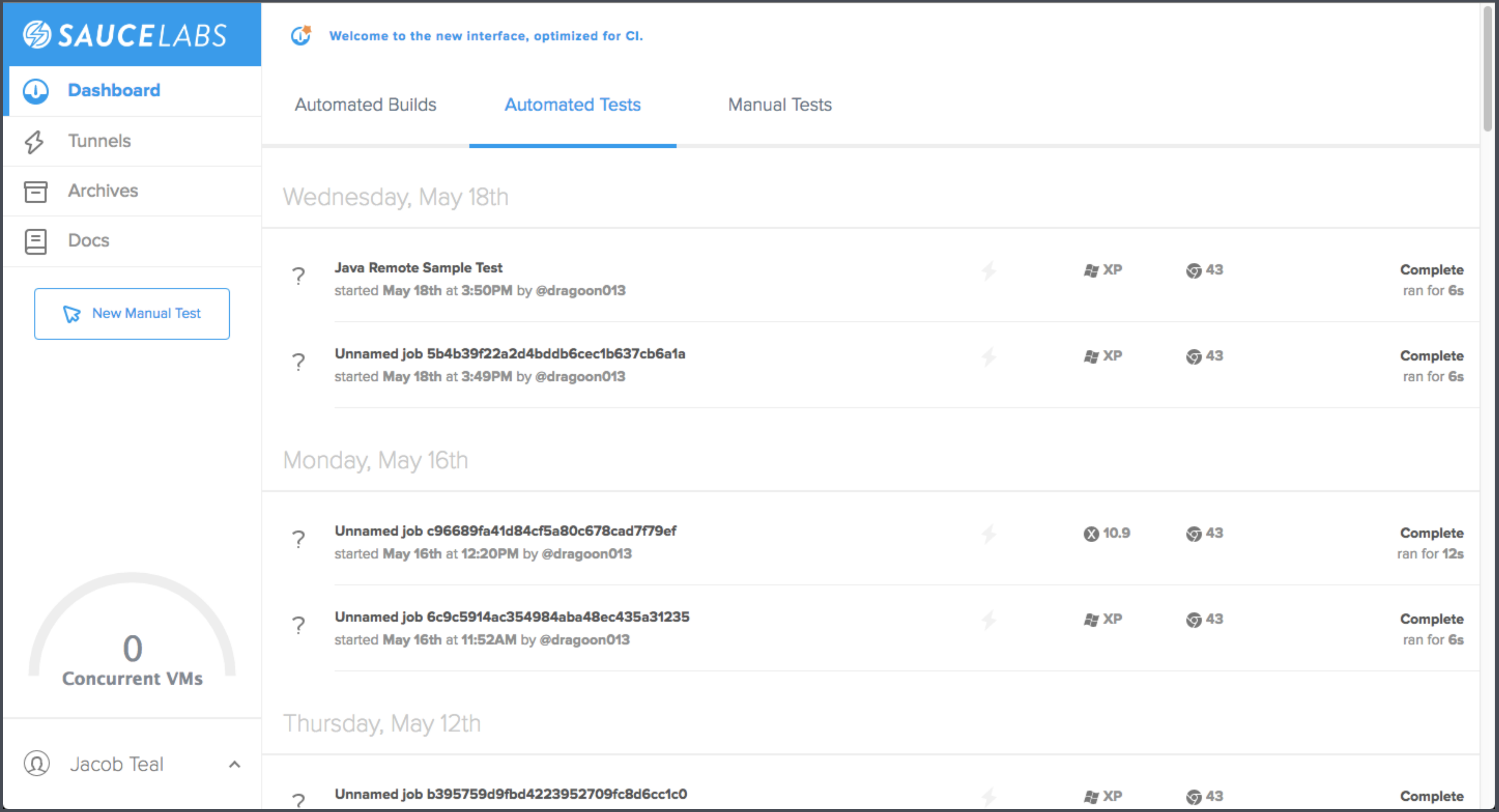
- Selenium Grid
- Parallelization
- Record results



Using Sauce Labs

- Selenium Grid
- Parallelization
- Automation
- Security
- Seamless Integration
- Recorded Tests

Sauce Labs UI



SAUCELABS

Welcome to the new interface, optimized for CI.

Wednesday, May 18th

?	**Java Remote Sample Test** started **May 18th** at **3:50PM** by **@dragoon013**	XP	43	**Complete** ran for **6s**
?	**Unnamed job 5b4b39f22a2d4bddb6cec1b637cb6a1a** started **May 18th** at **3:49PM** by **@dragoon013**	XP	43	**Complete** ran for **6s**

Monday, May 16th

?	**Unnamed job c96689fa41d84cf5a80c678cad7f79ef** started **May 16th** at **12:20PM** by **@dragoon013**	10.9	43	**Complete** ran for **12s**
?	**Unnamed job 6c9c5914ac354984aba48ec435a31235** started **May 16th** at **11:52AM** by **@dragoon013**	XP	43	**Complete** ran for **6s**

Thursday, May 12th

?	**Unnamed job b395759d9fbd4223952709fc8d6cc1c0**	XP	43	**Complete**

Manual Testing

The traditional method of testing applications and sites

- Testing Locally
- Sauce Labs Manual Tests

Documentation

Let's take a tour of the Docs!

- Sauce Labs Documentation
- Selenium Documentation
- Sauce Labs Sample Test Scripts
- Sauce Labs Sample Test Frameworks

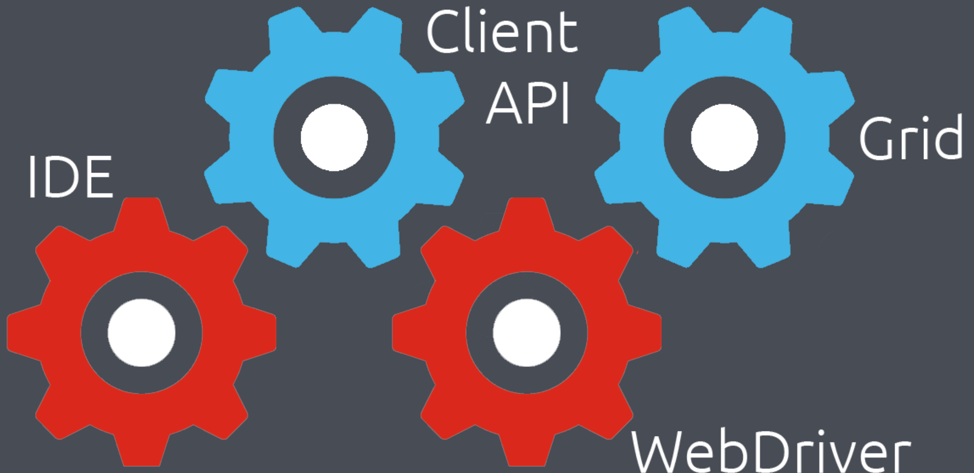
SELENIUM TEST SCRIPT BASICS

Module Objectives

This module enables you to:

- Understand how Selenium works
- Understand the basic components of a Selenium test script
- Run a Selenium test script locally

Cogs in the Selenium Machine



How Selenium WebDriver Works

1. Write script
2. Run script (send commands via API)
3. WebDriver Client sends commands to browser
4. WebDriver Client sends results back

7 Basic Steps of a Selenium Script

1. Create a Webdriver Object
2. Navigate to a Web Page
3. Locate HTML Elements on a Web Page
4. Perform an Action on HTML Elements
5. Anticipate Browser Response
6. Run the Test and Record the Test Results
7. Conclude the Test

1. Create a Webdriver Object

```
WebDriver driver = new FirefoxDriver();
WebDriver driver = new InternetExplorerDriver();
WebDriver driver = new ChromeDriver();
```

Remote (Sauce Labs) WebDriver

```
USERNAME = "dragoon013";
KEY = "secret";
URL = "https://" + USERNAME + ":" + KEY +
      "@ondemand.saucelabs.com/wd/hub";
WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
```

Desired Capabilities

```
DesiredCapabilities caps = new DesiredCapabilities();
           caps.setCapability("platform", "Windows XP");
           caps.setCapability("browserName", "Chrome");
           caps.setCapability("version", "43.0");
```

2. Navigate to a Web Page

```
driver.get("http://saucelabs.com/");
```

3. Locate HTML Elements on a Web Page

```
river.findElement(By.xpath("//*[@id=\"site-header\"]/div[3]/div/div/div[3]/div/span/button
```

Identify by:

- ID
- Class Name
- Tag Name
- Name
- Link Text
- Partial Linktext
- CSS
- XPath

Popular Locator Helpers

- Browser Developer Tools
    - Inspect the DOM
    - Firebug (Firefox)
    - Chrome developer tools
    - Internet Explorer developer tools

4. Perform an Action on a HTML Element

```
driver.findElement(By.name("q")).sendKeys("Hi");
```

5. Anticipate Browser Response

Implicit Waits

- A blanket wait time that applies to each findElement search (findElement(By.id())).

Explicit Waits

- A wait time set until a condition of an element (such as "selected" or "visible") becomes true.

6. Run the Test and Record the Test Results

- Print or save your test results locally, as part of of your script
- Test Frameworks
- Use the Sauce Labs API to send results to your Dashboard

- Record test results with Sauce Labs
  - Logs
  - Screencasts
  - Screenshots

7. Conclude the Test

```
driver.quit()
```

Lab 1: Running the Sample Script

1. Open Eclipse from the desktop shortcut.
2. In the package explorer left hand side panel, open java-testng-simple > src/test/java > com.yourcompany > SampleSauceTest.java.
3. Make a note of the website we tell the driver to access.
4. Right click on java-testng-simple and select Run As > 9 Maven test.

Lab 1: Running the Sample Script (cont.)

1. From the driver.get line, change the website to seleniumhq.org, then click Save.
2. Run the test again.
3. What happened this time?

WRITE A SELENIUM SCRIPT

Module Objectives

This module enables you to:

- Use different page actions in a Selenium test script

Identifying Elements

Locator Expressions are made in Key:Value pairs

```
WebElement linkElement = driver.findElement(By.id("i am a link"));
```

Locator Types

ID

```
driver.findElement(By.id("i_am_an_id"))
```

Tag Name

```
driver.findElements(By.tagName("a"))
```

Text

```
driver.findElements(By.linkText("i am a link"))
```

Locator Types continued

CSS Selector

```
driver.findElement(By.cssSelector("input#submit"))
```

XPath

```
driver.findElement(By.xpath("/html/head/title"))
```

Performing an Action

- Once you locate an element, you can perform an action on it
- Perform actions by invoking interaction methods on the WebElement object we identified earlier

`WebElement` `linkElement`

Click

Find and Act

```
driver.findElement(By.id("i am a link")).click();
```

Find, Store, and Act

```
WebElement linkElement = driver.findElement(By.id("i am a link"));
linkElement.click();
```

SendKeys

```
WebElement commentBox = driver.findElement(By.id("comments"));
commentBox.sendKeys("I think this is a pretty cool training site!");
```

Submit

```
WebElement submitButton = driver.findElement(By.id("submit"));
submitButton.submit();
```

Other Actions

- Clear
- Hover
- SwitchTo
- and more!

Lab 2: Writing Actions

1. Open SampleSauceTest.java in the java-testng-simple project.
2. If you have not done so, change the `driver.get("URL")` to `driver.get("http://saucelabs.github.io/training-test-page/")`.
3. Comment out the xpath and name element locator and corresponding action(s).
4. Open a new Chrome browser window and open the training test page site.
5. Right click on the Comments box, choose Inspect. Notice in the Page Inspector, that the comments box id is "comments".

Lab 2: Writing Actions (cont.)

1. In Eclipse, after `driver.get(URL)`, enter `driver.findElement(By.id(“comments”)).sendKeys(“Hello World”);` Then click Save.
2. Go back to the training page. Right click on the "send" button, choose Inspect. Notice in the Page Inspector, that the send button id is "submit".
3. In Eclipse enter `driver.findElement(By.id("submit")).submit();`
4. Uncomment and fill the while loop commands that locate and toggle **both** the checkboxes on the page (Inspect the checkboxes).
5. Save and run the script!

TESTING WITH SAUCE LABS

Module Objectives

This module enables you to:

- Run a Selenium test script on Sauce Labs

Testing Script Requirements

- Remote WebDriver
- Sauce Labs Authentication
- Desired Capabilities

Sauce Labs Authentication

Sauce labs authentication in your test script verifies the user and records test results against that user's profile

- Environment Variables

```
export SAUCE_USERNAME=dragoon013
export SAUCE_ACCESS_KEY=b89rmvitnbiq3rrr09u4r90fjr
```

- Explicit Variables

```
public static final String USERNAME = "dragoon013";
public static final String ACCESS_KEY = "b89rmvitnbiq3rrr09u4r90fjr";
```

Setting the Remote WebDriver

Sauce Labs API Authentication

- Username
- Access Key
- API URL

```
USERNAME = "dragoon013";
KEY = "b89b7205-3aca-4af2-a618-9bab4c7bdebd";
URL = "https://" + USERNAME + ":" + KEY + "@ondemand.saucelabs.com/wd/hub";
WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
```

Desired Capabilities

Required:

- Browser
- Platform
- Version

Optional:

- Selenium Driver Version
- Chrome Driver Version
- IE Driver Version

```
DesiredCapabilities caps = DesiredCapabilities.chrome();
caps.setCapability("platform", "Windows XP");
caps.setCapability("version", "43.0");
```

Lab 3: Sauce Labs Scripting

1. Open the SampleSauceReportScript.java script in the java-testng-simple_SL_report project.
2. Copy the global variables above the `main()` function. Paste it into our SampleSauceTest.java in the java-testng-simple project.
3. Add in your username to the USERNAME String variable.

Lab 3: Sauce Labs Scripting (cont.)

1. Log into saucelabs.com.
2. From the lower left corner, go to Username > My Account.
3. Scroll down to Access Key, click Show.
4. Enter your password, then click Authorize.
5. Copy your Access Key.
6. In Eclipse, paste your Access Key into the script for the value of the ACCESS_KEY String variable.

Lab 3: Sauce Labs Scripting (cont.)

1. Go to the Platform Configurator.
2. Select the Selenium API option.
3. Select a PC device.
4. Select a Windows 8 OS.
5. Select a Chrome version 50.0 browser.
6. Copy and paste the resulting Java code into your script, at the start of the main() function.
7. Comment out the old Chrome driver and System.setProperty() directive.
8. Create a new RemoteWebDriver object:

```
WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
```

Lab 3: Sauce Labs Scripting (cont.)

1. Run the script.
2. Open the Automated Tests tab on saucelabs.com.
3. Click the job and view the elements of your test in the video, the logs, and the metadata.

Explicit and Implicit Waits

Implicit Waits

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Explicit Waits

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement messageElement = wait.until(
                            ExpectedConditions.presenceOfElementLocated(
                            By.id("loginResponse")));
```

Lab 4: Testing Waits

1. Open a browser and go to http://the-internet.herokuapp.com/dynamic_loading.
2. Select Example 2: Element rendered after the fact, then click Start.
3. Inspect the "Hello World" text. What is it's id?
4. In Eclipse, open java-testng-simple_waits > src/test/java > com.yourcompany >SampleSauceWaits.java.
5. Fill in the global variables with your Sauce Labs username and access key.
6. Note the findElement that looks for the "finish" id.
7. Run the test. What happened and why?

Lab 4: Testing Waits (cont.)

1. Add in the Wait command for a 10 seconds wait.
2. Apply the wait to the findElement for "Finish":

   ```
   wait.until(ExpectedConditions.presenceOfElementLocated(By.id(“finish”)));
   ```
3. Run the script again.
4. Go to the Sauce Labs Dashboard and view your two tests, one with an error and the other complete.

INTRODUCTION TO TESTING FRAMEWORKS

Module Objectives

This module enables you to:

- Understand the use of Testing Frameworks
- Understand the use of asserts
- Use a framework to record test results in Sauce Labs

What is a Testing Framework?

A test automation framework is a scaffold comprised of libraries, dependencies, drivers, and helper scripts that facilitate the execution of Selenium test scripts.

Popular Frameworks

Java:

- TestNG
- JUnit

Ruby:

- RSpec
- Cucumber

Python:

- Robot
- UnitTest

JavaScript:

- Protractor
- Nightwatch

Test Driven Development vs. Behavior Driven Development

TDD: A developer writes the test, and then writes code until the test passes

BDD: A way to plan and develop code for particular features/scenarios that incorporates non-technical perspectives on the team

TDD Planning Example

Sauce_counter() needs to count the number of sauces inputted

Pasta_counter() needs to count the number of pastas inputted

sNpComparer() needs to compare the counts and make sure that they are equal

BDD Planning Example

Feature: Pasta and Sauce Matchmaker

Scenario: Some sauces

> Given I have 5 different kinds of Sauce
>
> When I have 5 different kinds of Pasta
>
> Then my chef will be happy

BDD Framework Example

```
@Given("^I have (\\d+) sauces for my pasta")
public void the_number_of_sauces(Int arg1) throws Throwable {
}

@When("^I have (\\d+) kinds of pasta")
public void the_number_of_pastas(Int arg1) throws Throwable {
}

@Then("^My chef will be (.*)$")
public void the_feeling_is(String arg1) throws Throwable {
}
```

BDD Frameworks

Python:

- Behave
- Lettuce

JavaScript:

- CucumberJS

Java:

- CucumberJVM

Ruby:

- Cucumber

Asserts:

A testing framework directive that assumes a value to be true, and stops the test and/or throws an error if false.

```
assertEquals(page.getUncheckedCheckBoxState(), true);
```

Assert Functions

- assertEquals
- assertNull
- assertSame

- assertThrows
- assertTrue
- assertEqualsNoOrd

Lab 5: Asserting Your Greatness

1. In Eclipse, go to java-testng-simpleSL-report > src/test/java > com.yourcompany > SampleSauceReportScript.java.
    - Notice the *assertEquals* line for `getTitle()`.
2. Run the test.
3. Notice the output in the Eclipse console. What happens to the test and why?
4. Correct the expected value that compares against the `getTitle()`, save, and run the test again.

Record your Results with Sauce Labs

- Use the update_job method in the Sauce Labs REST API after the test runs
- Use the Java Helper library; it will automatically send pass/fail results to Sauce Labs
- Configure your testing framework to send results to Sauce Labs

Lab 6: Record Keeping

1. Notice the String id of the session from Sauce Labs, the Sauce Labs API object, and the @AfterMethod function, after.
2. Uncomment the after function, and the line where the id variable value is set.
3. Run the test.
4. In the refreshed Sauce Labs Dashboard, notice the changes in how the pass criteria appear.

Lab 6: Record Keeping (cont.)

1. Go back in Eclipse to the script.
2. Change the if statement so that the test will pass.
3. Run the test.
4. In the refreshed Sauce Labs Dashboard, notice the indications for a successful test in the UI.

TESTING STRATEGY AND BEST PRACTICES

Module Objectives

This module enables you to:

- Understand testing strategies for setting up automated testing
- Label, name, and tag your tests to faciliate searching for past tests on Sauce Labs

Small, Atomic, and Autonomous Testing

Small: Tests should be short and succinct

Atomic: Tests should focus on testing a single feature.

Autonomous: Tests should be independent of other tests.

Types of Testing

- Unit Testing
- Functional Testing
- Performance Testing

Unit Testing

Unit tests are written from a developer persepective to ensure a method or function performs a set of tasks.

Functional Testing

Functional tests are written from a user perspective to test a functionality or feature of your application.

Performance Testing

Performance tests that gauge and output performance metrics for your application.

Test Dependencies

- Hard coding dependencies to access external account or data
- Test Set-Up
- Test Teardown

Setup

Setup initiates *prerequisite* tasks to be taken care of before your test runs, usually setting the capabilities, configuring additional browser parameters, and more.

Example:

```
public void setupLogin() {

    DesiredCapabilities caps = new DesiredCapabilities();
    caps.setCapability("platform", "Windows XP");
    caps.setCapability("browserName", "Chrome");
    caps.setCapability("version", "43.0");
}
```

Teardown

The teardown function includes *Post requisite* tasks that need to occur, like closing the browser, logging out, or terminating the remote session.

Example:

```
public void tearDown(boolean testTrue) throws Exception {

    webDriver.get().quit();

    if (testTrue)) {
        r.jobPassed(id);
    } else {
        r.jobFailed(id);
    }
}
```

Object Oriented Testing

Page Objects

- Code reuse across tests; reuse common element interactions
- Abstraction: With product change, only change one piece of code

Abstracting Page Element Locators

```java
@FindBy(id="i_am_a_textbox")
private WebElement textInput;

/*Set Page Object Actions*/

public static GuineaPigPage getPage(WebDriver driver) {
    return PageFactory.initElements(driver, GuineaPigPage.class);
}

public void enterCommentText(String text){
    this.TextInput.click();
    setTextInput(this.textInput, text);
}
```

Page Object Example: Incorrect

```
public class Login {

    public void testLogin() {
        driver.type("inputBox", "testUser");
        driver.type("password", "my supersecret password");
        driver.click("sign-in");
        driver.waitForPageToLoad("PageWaitPeriod");
        Assert.assertTrue(driver.isElementPresent("compose button"),
        "Login was unsuccessful");
    }
}
```

Page Object Example: Correct

```java
public class TestLogin {

    public void testLogin() {
        SignInPage signInPage = new SignInPage(selenium);
        HomePage homePage = signInPage.loginValidUser("userName", "password");
        Assert.assertTrue(driver.isElementPresent("compose button"),
        "Login was unsuccessful");
    }
}
```

Parallelization

Avoid dependencies between tests

- If you chain tests together, when one at the top fails, then they will all fail

Use Frameworks

- Frameworks include helpful libraries and functionality that can help you make the most of parallelization

Label Your Tests

- ID tests
- Name tests
- Apply build names to tests

```
caps.setCapability("tags","tag_awesome");
caps.setCapability("build","cool_builds1");
caps.setCapability("name","Java Remote Sample Test");
```

Docs: Test Configuration Options

Lab 7: Labelling and Naming our Tests

1. In Eclipse, open java-testng-simple > com.yourcompany > SampleSauceTest.java.
2. Add a capability to name your test.
3. Add a capability to include a build number for your test.
4. Add a capability to include a tag for your test.
5. Run the test script.
6. Open saucelabs.com. How does your test look different on the Automated Tests tab?

Best Practices Recap

- Keep Tests:
  - Small
  - Atomic
  - Autonomous
- Parallelization
- Abstracted Scripts
- Before and After methods

ADDITIONAL RESOURCES

Further Information

- SeleniumHQ: Documentation
- Dave Haeffner's Selenium Newsletter
- Sauce Labs Documentation
- Sauce Labs Sample Test Scripts
- Sauce Labs Sample Test Frameworks

Q&A

- Survey!
- Support: help@saucelabs.com