APPIUM 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
- Appium Server GUI
- Android Virtual Device (AVD)
- Android vs. iOS
- Saucelabs.com Account

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.

Preparing for the Labs

1. Double-click the Android_virtual_device.bat file on the desktop to start the AVD.
2. Open Eclipse from the desktop shortcut.
3. Double-click the Appium desktop icon.
4. After the device loads, press the play button on the Appium GUI.

Agenda

- Introduction to Sauce Labs and Appium
- Appium Basics
- Write Appium Test Scripts
- Appium Testing with Sauce Labs
- Introduction to Testing Frameworks
- Automated Testing Best Practices

INTRODUCTION TO SAUCE LABS AND APPIUM

Module Objectives

This module enables you to:

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

What is Sauce Labs?

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

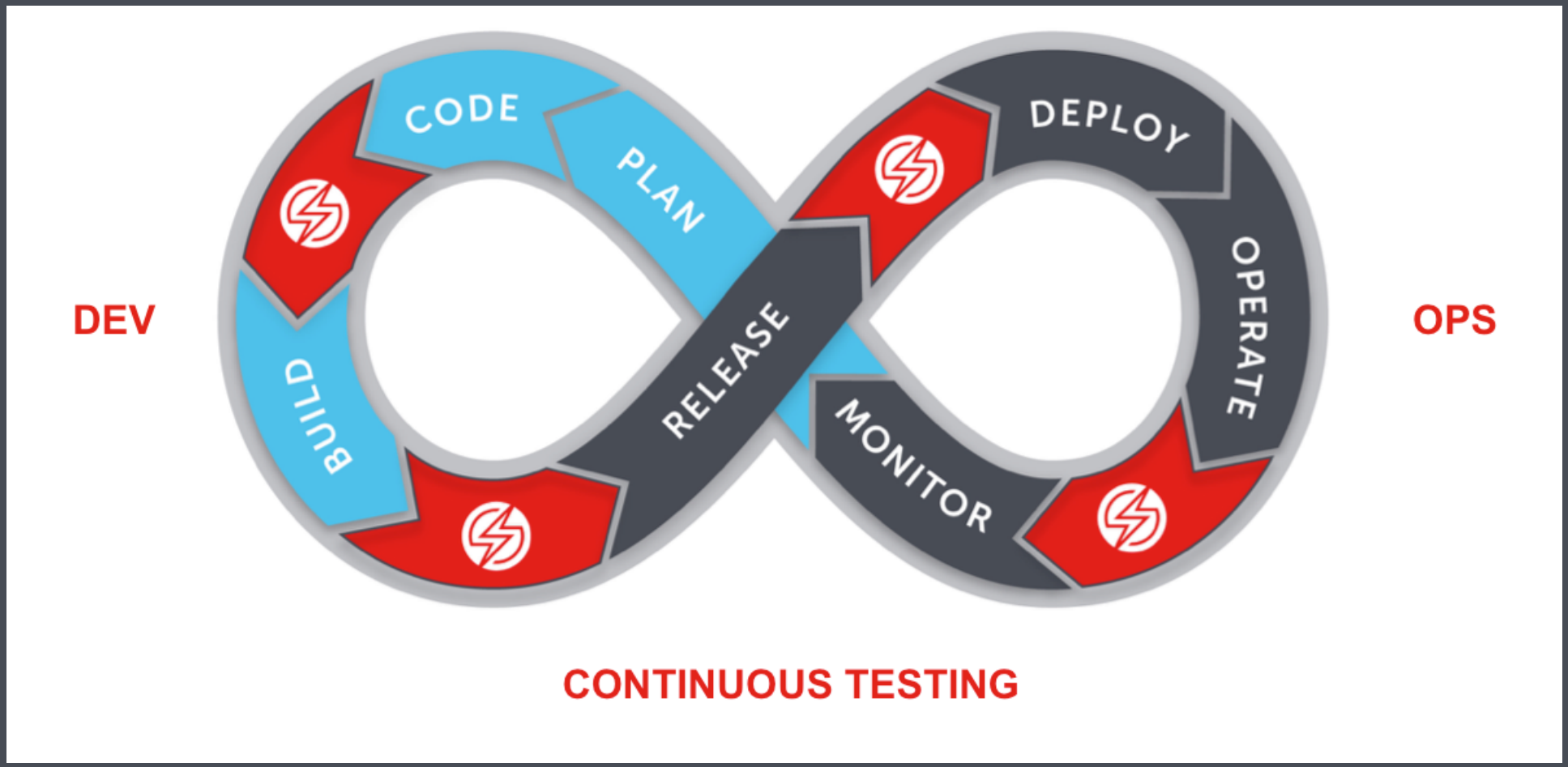
- Web Automated Testing
- Mobile Automated Testing

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 Appium?

Open source mobile application automation framework developed in 2011 by Dan Cuellar, the Test Manager at Zoosk.

Uses UIAutomation Framework to run in real time like an interpreter.

- Simulate mobile gestures
- Automate Mobile Application Testing

Local vs. Sauce Labs Appium Tests

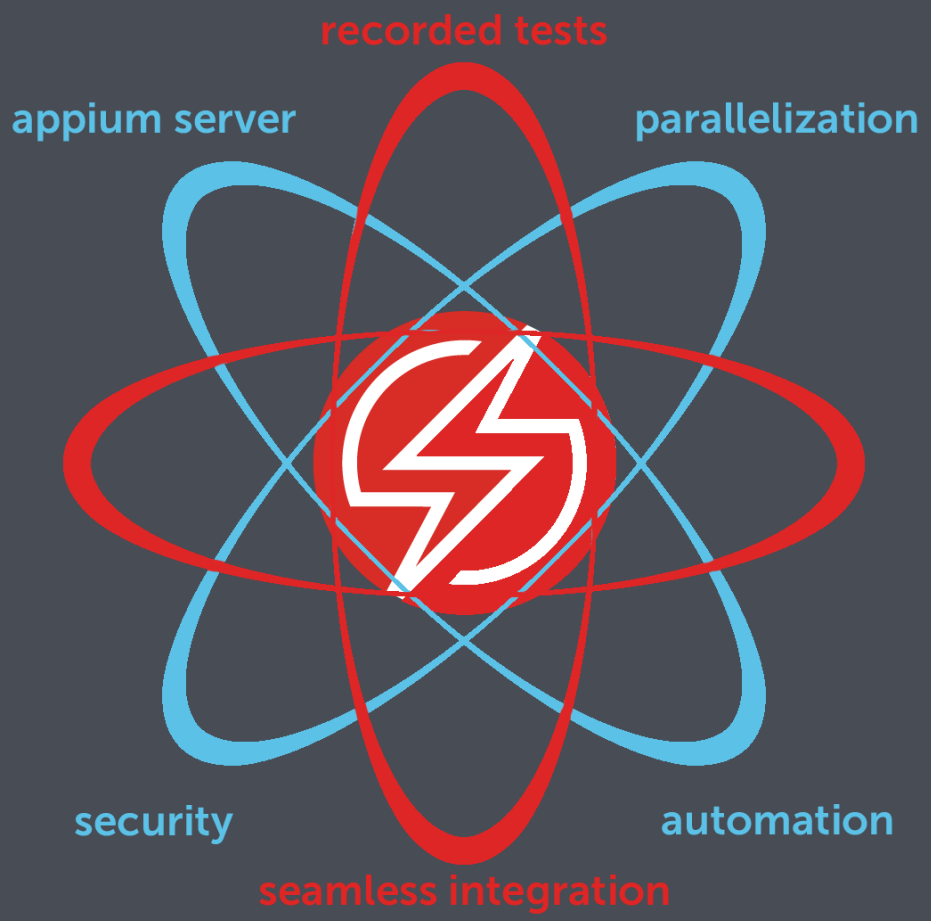
Local:

- Requires Appium Server running locally or remotely
- iOS development requires Xcode tools
- Need all the libraries and images to run device simulators/emulators

Sauce Labs:

- Appium server is remote and provisioned for you
- Real and virtual devices provisioned
- Parallelization
- Record Results

Using Sauce Labs



Sauce Labs UI

SAUCELABS

Dashboard
Tunnels
Archives
Docs

New Manual Test

0
Concurrent VMs

Jacob Teal

Welcome to the new interface, optimized for CI.

Automated Builds | Automated Tests | Manual Tests

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

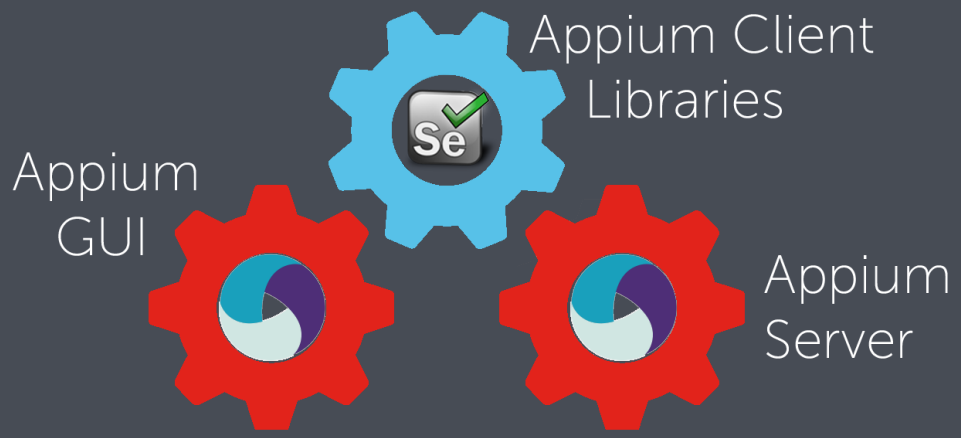
APPIUM BASICS

Module Objectives

This module enables you to:

- Understand how Appium works
- Understand the basic components of Appium test scripts
- Run an Appium test script on a local emulator

Appium Nuts and Bolts



- Android UIAutomator and iOS UIAutomation Frameworks
- Appium Client Libraries (extends Selenium Webdriver)
- Appium Server
- Appium GUI

UI Automator (Android)

- Uses Java to interact with the UI Automator API
- Developed by Google as an Automation framework for Android application testing

Documentation: Android UI Automator

UI Automation (iOS)

- Uses JavaScript to interact with the UI Automation API
- Developed by Apple as an Automation framework for iOS application testing

Documentation: iOS UI Automation

Testing on Windows Operating Systems

Requirements:

- Node.js
- Appium.exe
- Android SDK
  (>= API Level 17)

)

- Java
- Apache Ant
- Apache Maven

Documentation: Running Appium on Windows

Testing on Mac OS X Operating Systems

Requirements:

- Requires Mac OS X
- Xcode >= 7.1 recommended
- iOS SDK(s)
- Authorize iOS

Documentation: Running Appium on Mac OS X

Types of Mobile Apps to Test

- Mobile Web Browser Application
- Native Application
- Hybrid Application

The Appium Process

1. Write script
2. Run script - Sends commands to Appium Server via Appium Client library
3. Appium Server sends commands to mobile simulator/emulator
4. Appium Server sends results back
5. (On Sauce Labs) Test Assets are created

Basic Steps of an Appium Script

1. Set the Desired Capabilities (or Set the Application Path)
2. Start an Appium Session
3. Locate an Element in the Application
4. Perform an Action on an Application Element
5. Anticipate Application Response
6. Validate Responses and Reporting Test Results
7. Conclude a Test

Lab 1: Running the Sample Appium Script

1. If you haven't already, start the Appium Server.
2. In the Eclipse package explorer left panel, local-java-testng-appium-android > src/test/java > com.yourcompany > Local_SampleSauceTest.java.
3. Note the identified WebElement, and what we are doing to it.
4. Right click java-testng-simple and select Run As > 9 Maven test.
5. Click the device and Appium icons on the toolbar.
6. What did you see happen? How did it happen?

WRITE AN APPIUM TEST SCRIPT

Module Objectives

This module enables you to:

- Identify elements in your application structure
- Write actions into your Appium test script
- Run a local Appium Android test

Desired Capabilities

Describes a series of key/value pairs that encapsulate aspects of a browser.

Web Browser

```
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("platformVersion", "4.4");
capabilities.setCapability("deviceName", "Android Emulator");
capabilities.setCapability("browserName", "Chrome");
```

Native/Hybrid

```
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("platformVersion", "4.4");
capabilities.setCapability("deviceName", "Android Emulator");
capabilities.setCapability("app", "https://github.com/appium/app.apk");
```

Appium and Mobile Drivers

Android

```
AndroidDriver<MobileElement> driver = new AndroidDriver<MobileElement>(
    new URL("http://localhost:4723/wd/hub"),
    capabilities);
```

iOS

```
IOSDriver<MobileElement> driver = new IOSDriver<MobileElement>(
    new URL("http://localhost:4723/wd/hub"),
    capabilities);
```

Different Drivers

- RemoteWebDriver
- AppiumDriver
- AndroidDriver
- IOSDriver

Identifying Elements

Locator Expressions are made in Key:Value pairs

```
WebElement newGameBtn = driver.findElement(
    By.id("com.google.android.divideandconquer:id/newGame"));
```

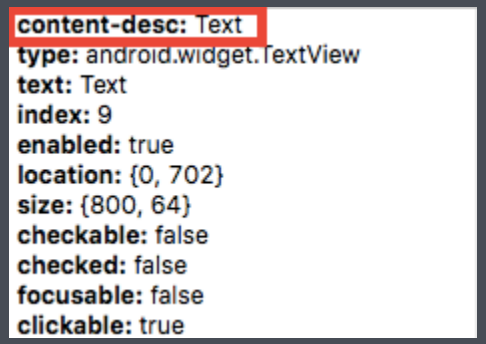
Android Locators

UI Automator class name

```
textView = driver.findElementByClassName("android.widget.TextView");
```

Acessibility id

```
textBox = driver.findElementByAccessibilityId("Text");
```



Android Locators cont.

-android uiautomator

- UIAutomator language

```
UiDevice device = UiDevice.getInstance(getInstrumentation());
UiObject cancelButton = device
    .findObject(new UiSelector()
    .text("Cancel"))
    .className("android.widget.Button"));
```

- Appium script

```
WebElement cancelButton = driver.findElementsByAndroidUIAutomator(
     "new UiSelector().text(\"Cancel\").className(\"android.widget.Button\")");
```

iOS Locators

UI Automation class name

```
MobileElement fieldTwo = (MobileElement) driver.
    findElementsByClassName("UIATextField").get(1);
```

Acessibility id

```
MobileElement fieldOne = (MobileElement) driver.
    findElementByAccessibilityId("TextField1");
```

iOS Locators cont.

-ios uiautomation

- UIAutomation JavaScript API script

```
UIATarget.localTarget().frontMostApp().mainWindow()
    .scrollViews()[0].cells()
    .firstWithPredicate("name matches 'Sign In'");
```

- Appium script (Java)

```
String uiautoFind = "UIATarget.localTarget().frontMostApp().mainWindow()
    ".scrollViews()[0].cells()
    .firstWithPredicate(\"name matches 'Sign In'\")";
driver.findElementByIosUIAutomation(uiautoFind);
```

Lab 2: Getting the Page Source

1. Restart the Appium Server.
2. Bring Eclipse to the foreground.
3. In Local_SampleSauceTest.java, write in the following line of code in the line before the `tap()` action.

```
System.out.println(driver.getPageSource());
```

4. Run the test.
5. Note the XML that apears in the Eclipse console.

Lab 2: Analyzing the Page Source

1. Go to your desktop and open the folder XML Inspector Examples.
2. Double-click on GraphicsViewScreen.xml.
3. Note the formatted XML. Can you recognize elements in the XML that appear in the application?

Mobile Specific Actions (Gestures)

- Once you locate an element, you can perform an action on it
- Perform actions by invoking the driver, and then a gesture action that is performed on an element

Touch Actions

There is a TouchAction class in the appium client libraries, which contain the different actions you can perform on a mobile device.

```
TouchAction touch = new TouchAction(driver);
```

Gesture Actions

press(x, y)

```
//press(org.openqa.selenium.WebElement el, int x, int y)
touch.press(btnElem, 150, 100).perform();
```

moveTo(+x, +y)

```
//moveTo(org.openqa.selenium.WebElement el, int x, int y)
touch.press(10, 200).moveTo(1, 1).perform();
```

tap(element)

```
//tap(org.openqa.selenium.WebElement el, int x, int y)
touch.tap(btnElem).perform();
```

Other Gesture Actions

waitAction(int ms)

```
//waitAction(int ms)
action1.press(element1).waitAction(300).moveTo(10, 0).release().perform();
```

longPress(x, y, duration)

```
//longPress(org.openqa.selenium.WebElement el, int x, int y, int duration)
action.longPress(element1).moveTo(x1, 580).release().perform();
```

perform()

```
action.longPress(element1).moveTo(x1, 580).release().perform();
```

MultiTouch

```
MultiTouchAction multiAction = new MultiTouchAction(driver);

TouchAction finger1 = new TouchAction(driver);
TouchAction finger2 = new TouchAction(driver;

finger1.longPress(x1, y1).waitAction(1500);
finger2.longPress(x2, y2).waitAction(1500);

multiAction.add(finger1).add(finger2).perform();
```

Switching Contexts

```
Set<String> contextNames = driver.getContextHandles();

for (String contextName : contextNames){
    if (context == "NATIVE_APP")
        driver.switchTo().window("WEBVIEW_1");
    } else if(context == "WEBVIEW_1"){
        driver.switchTo().window("NATIVE_APP");
    }
}
```

Explicit and Implicit Waits

Implicit Waits

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

Explicit Waits

```
//additional libraries required
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

//usage
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement messageElement = wait.until(ExpectedConditions.
        presenceOfElementLocated(By.id("loginResponse")));
```

Lab 3: Action!

1. Restart the Appium Server.
2. In Local_SampleSauceTest.java, add in a `scrollTo()` below our `tap action`.

```
driver.scrollTo("Touch Paint").click();
```

Lab 3: Actions! cont.

1. Uncomment the three TouchAction objects (eye1, eye2, smile).
2. Uncomment the `smile.press()` column of `moveTo()` actions.
3. Below the TouchAction objects, insert a line that initiates a press at x=150 and y=200 (150,200).
4. Add an action to release the touch at the end of the press.
5. Add a `perform()` action to the end of the press and release sequence.
6. Repeat steps 3-5 at x=250 and y=200.
7. Save the script and run the test.

Lab 3: Waiting...

1. Restart the Appium Server.
2. Implement a wait to give the Touch Paint activity time to load.
3. Below the AndroidDriver object, write an explicit WebDriver wait, that waits for 10 seconds.
4. Uncomment the `wait.until()` line.
5. Save the script and rerun the test.
6. What do you see on your application screen?

APPIUM TESTING WITH SAUCE LABS

Module Objectives

This module enables you to:

- Run an Appium test script on Sauce Labs

Sauce Labs Appium Script Pre-Reqs

- Use Sauce Labs storage or another online location to host the application
- 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="YOUR_SAUCE_USERNAME"
export SAUCE_ACCESS_KEY="YOUR_ACCESS_KEY"
```

- Explicit Variables

```
public static final String USERNAME = "YOUR_SAUCE_USERNAME";
public static final String ACCESS_KEY = "YOUR_ACCESS_KEY";
```

Sauce Labs Authentication, con't.

Sauce Labs Authentication

- Username
- Access Key
- API URL

```
public static final USERNAME = "YOUR_SAUCE_USERNAME";
public static final KEY = "YOUR_ACCESS_KEY";
public static final URL = "https://" + USERNAME
                        + ":" + KEY + "@ondemand.saucelabs.com/wd/hub";

@Test
public static void main(){
    DesiredCapabilities capabilities = new DesiredCapabilities();
    //additional capabilities
    AndroidDriver<MobileElement> driver = new AndroidDriver<MobileElement>
                                          (new URL(URL), capabilities);
```

Desired Capabilities

Required:

- platformName
- platformVersion
- deviceName
- app
- browserName
- appiumVersion

Optional:

- deviceOrientation
- appActivity

```
DesiredCapabilities caps = new DesiredCapabilities();

capabilities.setCapability("platformName", "Android");
capabilities.setCapability("platformVersion", "4.4");
capabilities.setCapability("browserName", "");
capabilities.setCapability("appiumVersion", "1.5.3");
capabilities.setCapability("deviceName", "Android Emulator");
capabilities.setCapability("app", "https://github.com/appium/app.apk");
```

Application Hosting

- Sauce Storage
- Locally hosted webserver
- Remote host: AWS, DigitalOcean, Github, etc

```
curl -u <sauce_username>:<sauce_access_key> \
    -X POST -H "Content-Type: application/octet-stream" \
    https://saucelabs.com/rest/v1/storage/<sauce_username>/<upload_filename>?overwrit
    --data-binary @<path/to/myApp.apk>
```

```
capabilities.setCapability("app", "sauce-storage:myApp.apk");
```

Lab 4: Testing Sauce Labs

1. Open a browser and log into saucelabs.com.
2. Go to My Account from the lower left drop-up menu and copy your access key.
3. In the Eclipse package explorer left panel, open remote-java-testng-appium-android > src/test/java > com.yourcompany > Remote_SampleSauceTest.java.
4. Copy your access key into the script for the `ACCESS_KEY` variable.
5. Fill out the `USERNAME` variable with the correct value for your account.

Lab 4: The Platform Configurator

1. Note the DesiredCapabilities object in Remote_sampleSauceTest.java.
2. Open a browser to the Platform Configurator shortcut.
3. Set the API to Appium.
4. Set the Device to Android Emulator Phone.
5. Set the Operating System to Android 4.4.
6. Set the Appium Version to 1.4.16.
7. Set the Web or App Testing to App Testing
8. Copy and paste the missing required lines into your script.

Lab 4: Running the Script on Sauce Labs

1. Save and run the script.
2. Open the Automated Tests tab on saucelabs.com.
3. What do you notice?
4. What after-test options are available?

INTRODUCTION TO TESTING FRAMEWORKS

Module Objectives

This module enables you to:

- Understand the use of Testing Frameworks
- Understand the use of asserts in Appium scripts
- 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, dependecies, drivers, and helper scripts that facilitate the execution of Appium test scripts.

- Parallelization
- Assertions
- Reporting

Popular Frameworks

Java:

- TestNG
- JUnit

Python:

- Robot
- UnitTest
- Nose

Ruby:

- RSpec
- Cucumber

JavaScript:

- Protractor
- Nightwatch
- Mocha

Test Driven Development vs. Behavior Driven Development

TDD: A developer writes the test, and then writes code, and tests that code continuously, until the test passes.

BDD: A way to plan and develop code similar to TDD, but allows teams to include non-technical members when writing tests for scenarios and features

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

Java:

- CucumberJVM

Python:

- Behave
- Lettuce

Ruby:

- Cucumber

JavaScript:

- CucumberJS

Asserts

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

```
assertTrue("The button is not visible", compassBtn.isDisplayed());
```

Assert Functions

- assertEquals
- assertNull
- assertSame
- assertThrows
- assertTrue
- assertEqualsNoOrd

Documentation: Assert Class details

Lab 5: Assertions

1. Note that the title of the Touch Paint activity is Graphics/Touch Paint.
2. Write an assertEquals assertion that uses the `getText()` function on the title WebElement object.
3. The assertEquals assertion should compare the value of `title.getText()` to the expected value, namely "Graphics/Touch Paint"
4. Save the script and run the test.

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. Open the Remote script for editing.
2. Uncomment the `id` variable under the AndroidDriver line.
3. Uncomment the AfterMethod function at the bottom of the script.
4. Change the expected string in the assertEquals to "Graphics/Touch Banana".
5. Save the script and run the test.
6. Open the saucelabs.com UI. What do your test results look like?

TESTING STRATEGY AND BEST PRACTICES

Module Objectives

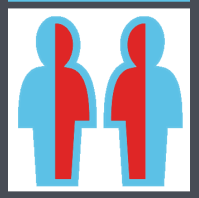
This module enables you to:

- 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.

Real vs. Simulator/Emulator Testing

Real

- Test hardware
- Test speed fluctuates
- Expensive
- Testing on real device
- Difficult to set up

Simulator/Emulator

- Consistently reproducible
- Parallelization
- Less expensive
- Easier to set up and do on your own
- Setting language, location, turn off popup blocking, etc.

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:

```
@BeforeMethod
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:

```
@AfterMethod
public void after(ITestResult testResult) {
    SauceREST restAPI = new SauceREST(USERNAME, ACCESS_KEY);

    if (testResult.isSuccess()) {
        restAPI.jobPassed(id);
    } else {
        restAPI.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
- Cross-platform tests: abstract your application element objects that exist across iOS and Android

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

Debugging

What to look for:

- Errors in your app, Sauce Labs UI, Appium Server
- Look at the test output, Appium logs, logcat, Android or iOS logs
- Use the pageGetSource() command to see struture of app at that moment of error

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: Labeling and Naming Your Tests

1. Open the Remote script for editing.
2. Add a capability to name your test.
3. Add a capability to include a build number for your test.
4. Run the test script.
5. Open saucelabs.com. How does your test look different on the Automated Tests tab? The builds tab?

ADDITIONAL RESOURCES

Further Information

- Appium Documentation
- Appium Issue Tracker
- Appium Sample Test Scripts
- Sauce Labs Documentation
- Sauce Labs Sample Test Scripts
- Sauce Labs Sample Test Frameworks

Q&A

- Survey!
- Support: help@saucelabs.com