Sauce Labs Enterprise: Course Labs

Lab 1: Sauce Connect Proxy

Goal Run a single manual tunnel over Sauce Connect Proxy

Time 10 minutes

Step	Action
1.	Ensure that Sauce Connect Proxy is installed on your machine. Follow the download and install instructions here Create two environment variable for your SauceLabs.com user and access key. • Right-click the start button • Click on "System" • Click on "Advanced System Settings" • Click on "Environment Variables" • Click on "New Variable" • Add SAUCE_USERNAME and 'your name' as the value • Add SAUCE_KEY and copy your access key from saucelabs.com, then enter it as the value.
2.	Open up your windows terminal. Run the following commands: • `C:\Users\SauceTraining\> cd Desktop\SC_Proxy\bin\` • `C:\Users\SauceTraining\Desktop\SC_Proxy\bin\> sc.exe -u %SAUCE_USERNAME% -k %SAUCE_KEY% -i readytech_tunnel`
3.	Run a manual test in SauceLabs and select the **`readytech_tunnel`** as your tunnel

Lab 2: Pre-Run Executable

Goal Create a HashMap that will download a bash script before a test run

Time 15 minutes

Step	Action
1.	Upload the disable_fraud.sh (on the desktop) script to sauce storage using the following curl command: `curl –u %SAUCE_USERNAME%:%SAUCE_KEY% -X POST -H "Content-Type: application/octet-stream" https://saucelabs.com/rest/v1/storage/%SAUCE_USERNAME%/disable_fraud.sh?overwrite=true --data-binary @/disable_fraud.sh`
2.	Open Eclipse, then open java-testng-simple
3.	Change the variables `USERNAME` and `ACCESS_KEY` to your Sauce Labs credentials.
4.	Add a public Hashmap named 'prerun' below the 'id' variable: `Public HashMap <String, String> prerun;`
5.	Above the `@Test` annotation add a `@BeforeTest` annotation
6.	Add a function called `before()` that references the hashmap and declares the prerun arguments, including the file you uploaded to sauce-storage `public void before() { prerun = new HashMap<String, String>(); prerun.put("executable","sauce-storage:disable_fraud.sh"); prerun.put("args", " -a" + " -q"); prerun.put("background","false"); System.out.println("Pretest assets set: " + prerun); }`
7.	In the `main()` function, add a "`prerun`" desired capabilities that references `prerun` as an executable. `caps.setCapability("prerun", prerun);`
8.	Save the script, and then run the test

	• Check the recording of the test to see the bash script being downloaded and executed before the test.
9.	Add the `/S` flag in the `prerun()` parameters. Save and run the test again and notice that the Sauce logs don't log as many console messages.

Lab 3: The Sauce REST API

Goal Use the SauceREST API to:

- List account names
- Get test activity for a given user
- Stop a test for a given user
- Get active tunnel information

Time 15 minutes

Step	Action
1.	Ensure you have a Sauce Connect Proxy tunnel instance running. `>sc.exe -u %YOUR_USERNAME% -k %YOUR_ACCESS_KEY% -i <id>`
2.	In SauceLabs.com, run a manual test against this URL: https://saucelabs.github.io/training-test-page/
3.	In your command terminal, use the Account API method to get a list of the running accounts of your profile: `> curl https://saucelabs.com/rest/v1/users/%YOUR_USERNAME% -u %YOUR_USERNAME%:%YOUR_ACCESS_KEY%`
4.	Use the activity REST API to get the current activity of running tests via a given user. `> curl https://saucelabs.com/rest/v1/%YOUR_USERNAME%/activity -u %YOUR_USERNAME%:%YOUR_ACCESS_KEY%`
5.	Use the job REST API to stop the currently running test via the Job ID `> curl -u %YOUR_USERNAME%:%YOUR_ACCESS_KEY% -X PUT -d https://saucelabs.com/rest/v1/%YOUR_USERNAME%/jobs/YOUR_JOB_ID/stop`

Lab 4: REST API in Your Test Script

Goal Implement the Sauce REST client library binding in your test script.

Time 10 minutes

Step	Action
1.	In Eclipse, open **SampleSauceRestTest.java**
2.	Create a public SauceREST variable called `restAPI`, a public String called `myJob`, and a public String called `tunnelID`
3.	Add a *`@BeforeTest`* annotation, followed by a public method called `before()`
4.	In `before()`, add new declaration for `restAPI` that uses *`USERNAME`* and *`ACCESS_KEY`* as parameters. `restAPI = new SauceREST(USERNAME, ACCESS_KEY);`
5.	Add a declaration for `tunnelID` that uses the `getTunnels()` method. `tunnelID = restAPI.getTunnels();`
6.	Finally, add a `System.out` to print the value of `tunnelID`. `System.out.println("Tunnels: " + tunnelID);`
7.	Run the Maven test and view the console messages in Eclipse. • Note: the value returned by tunnelID is not the value required for the 'tunnel-identifier' desired capability.

8.	Add a new declaration in the WebDriver section the uses the `getJobInfo()` REST API that returns the Job Details
	
	`myJob = restAPI.getJobInfo(id);`
	`System.out.println(myJob);`

9.	Lastly, uncomment the *`@AfterMethod`* to send an update job API call to view whether the test passed or failed.

Lab 5: High-Availability Sauce Connect

Goal Configure a pool of shared tunnels

Time 5 minutes

Step	Action
1.	Run 3 tunnels simultaneously by opening three command prompt tabs and enter the following commands: • `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc1.pid --logfile /tmp/sc1.log --scproxy-port 29997 --se-port 4446 -i my-tun1` • `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc2.pid --logfile /tmp/sc2.log --scproxy-port 29998 --se-port 4447 -i my-tun2` • `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc3.pid --logfile /tmp/sc3.log --scproxy-port 29999 --se-port 4448 -i my-tun3`
2.	Confirm that all tunnels are currently running in SauceLabs.com
3.	Open the SampleParrallelTests.java and edit "tunnel-identifier" values for each test (i.e. `tun1`, `tun2`, `tun3`)
4.	Run a maven test and see the tests running parallel with Sauce Connect
5.	Teardown the tunnels by using Ctrl + C in each command prompt OR by deleting them in the SauceLabs.com interface
6.	Run 3 tunnels again using the same commands as step 1, but give each tunnel the same id "`pool-tunnel`" and add the following flags: • `--no-remove-colliding-tunnels` • `--wait-tunnel-shutdown` ○ `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc1.pid --logfile /tmp/sc1.log --scproxy-port 29997 --se-port 4446 -i pooled-tunnel --no-remove-colliding-tunnels --wait-tunnel-shutdown` ○ `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc2.pid --logfile /tmp/sc2.log --scproxy-port 29998 --se-port 4447 -i pooled-tunnel --no-remove-colliding-tunnels --wait-tunnel-shutdown` ○ `sc.exe –u %SAUCE_USERNAME% -k %SAUCE_KEY% --pidfile /tmp/sc3.pid --logfile /tmp/sc3.log --scproxy-port 29999 --se-port 4448 -i pooled-tunnel --no-remove-colliding-tunnels --wait-tunnel-shutdown`

7.	Back in Eclipse, change the "tunnel-identifier" values again to "`pooled-tunnel`"
8.	Save and run the test again, what do you notice in SauceLabs.com?

Lab 6: Configure Sauce OnDemand

Goal Send the Sauce Labs a pass or fail

Time 5 minutes

Step	Action
1.	Create a new project on Jenkins (should be listening on localhost:8080) • User: admin • Password: password
2.	Choose Configure in Jenkins Build
3.	Change the Source Code Management to this address: https://github.com/saucelabs-training/Java-TestNG-Selenium-Jenkins •
4.	Enable SauceLabs Support in the Build Environment
5.	Choose at least two platforms for desired capabilities in the Sauce Labs Options
6.	Enable Sauce Connect checkbox
7.	Invoke a top-level Maven targets as another build step. For example Maven version 3.3.9, Goals = clean, test;
8.	Set the Test Publisher as a post action build step
9.	Run the build in Jenkins, then view the test results in SauceLabs.com

Lab 7: Sauce On-Demand Browsers

Goal Grab ENV variables within Jenkins and iterate and parse into JSON format.

Time 5 minutes

Step	Action
1.	Open project: java-testng-ondemand
2.	Scroll down to the @DataProvider annotation
3.	Construct a String variable to represent your JSONArray. Have the variable pull the system variable for "SAUCE_ONDEMAND_BROWSERS" `String browsersJSONArrayString = System.getenv("SAUCE_ONDEMAND_BROWSERS");`
4.	Create the JSONArray and pass the previous string value as a parameter `JSONArray browsersJSONArrayObj = new JSONArray(browsersJSONArrayString);`
5.	Create an Object (to represent the browser objects being passed through sauceBrowserDataProvider) and set the length to()[3] `Object[][] browserObjArray = new Object[browsersJSONArrayObj.length()][3];`

6.	Construct a for loop and iterate through the JSON Array, and parse each object as a JSON
	
	`for (int i=0; i<browsersJSONArrayObj.length(); i++) {`
	`JSONObject browserObj = (JSONObject)browsersJSONArrayObj.getJSONObject(i);`
	`browserObjArray[i] = new Object[]{browserObj.getString("browser"), browserObj.getString("browser-version"), browserObj.getString("os")};`

7.	Finally, ensure your Object returns the browser Object Array `return browserObjArray;`
8.	Save and Run your test as a Maven test
9.	What do you notice when you check SauceLabs.com and Jenkins on localhost:8080?