ENSE 375 Group Project Part #4 Issues

Jacob Sauer
5 April 2021

Selenium

Selenium is an open-source framework for performing system tests on web applications. Tests can be written natively in the language of an application, including Java by virtue of the org.openqa.selenium package, and executed on said application using any development environment.

We were able to install and configure Selenium to our Jenkins servers relatively painlessly, as evidenced by the deprecation-related issues that Snyk detected in Selenium v3.5. However, we encountered numerous compilation errors in the process of writing test cases with the Selenium API, and were ultimately unable to resolve them. Initially, we attempted to use the FirefoxDriver class in order to deploy test results to Mozilla Firefox, but we were never able to configure the path to the Firefox binary such that Jenkins would recognize it. Consequently, we replaced FirefoxDriver with ChromeDriver, and attempted to "brute-force" certain downloads to verify that the extra components were in their correct locations.

Below is an overview of our step-by-step process and explanations of our most prevalent error messages:

1. On the Jenkins dashboard, navigate to *Configure Jenkins –> Manage Plugins*.
2. Install "Selenium" from the *Available* tab.
3. Update *pom.xml* to include a particular version of Selenium among the project's dependencies:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <scope>test</scope>
        </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.141.0</version>
    </dependency>
</dependencies>
```

4. In *AddPatientValidTestCases.java*, write a new test case that employs a basic function of the Selenium API, which is retrieval of a webpage URL.

```
@Test
public void TestSelenium() throws Exception
{
        File chromeDriver = new File("chromedriver");
        chromeDriver.setExecutable(true);
        System.setProperty("webdriver.chrome.driver", "chromedriver");
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--no-sandbox");
        options.addArguments("--disable-dev-shm-usage");
        WebDriver driver = new ChromeDriver(options);
        driver.get("http://demo.guru99.com/test/guru99home/");
        String title = driver.getCurrentUrl();
        assertEquals("http://demo.guru99.com/test/guru99home/", title);
}
```

5. Build the Jenkins project.

At this point, we expected to witness a successful build in which ChromeDriver would open the above URL in GoogleChrome and return it to the Java script. Instead, we encountered several compiler errors related to the behaviour of ChromeDriver and Google Chrome, and although some errors were resolved, others persisted for the duration of the project. Examples of such errors are listed below:

org.openqa.selenium.WebDriverException: ChromeDriver is not executable.

Initially, the Jenkins project would not compile because the ChromeDriver file that was pulled from our GitHub repository to the Jenkins slave was not an executable. We initially attempted to resolve this with the line *chromeDriver.setExecutable(true);*, but later re-uploaded it as an executable file. The latter action resolved this error.

org.openqa.selenium.WebDriverException: Chrome binary not found.

In hindsight, this was a rather simple error to resolve, although it did require multiple actions to be taken. To begin, the version of Google Chrome that we initially uploaded to our GitHub repository was incompatible with the Jenkins slave's operating system. Consequently, it was necessary to enter the Docker container as the root user, download the latest stable release of Google Chrome for Debian, and also upload it to GitHub. Unfortunately, this download alone did not resolve the issue; it was additionally necessary to verify that a path to the Google Chrome binary existed in the path */usr/bin/google-chrome*. This action ultimately mitigated the error.

org.openqa.selenium.WebDriverException: Failed to launch the browser process: error while loading shared libraries: libnss3.so: cannot open shared object file: No such file or directory.

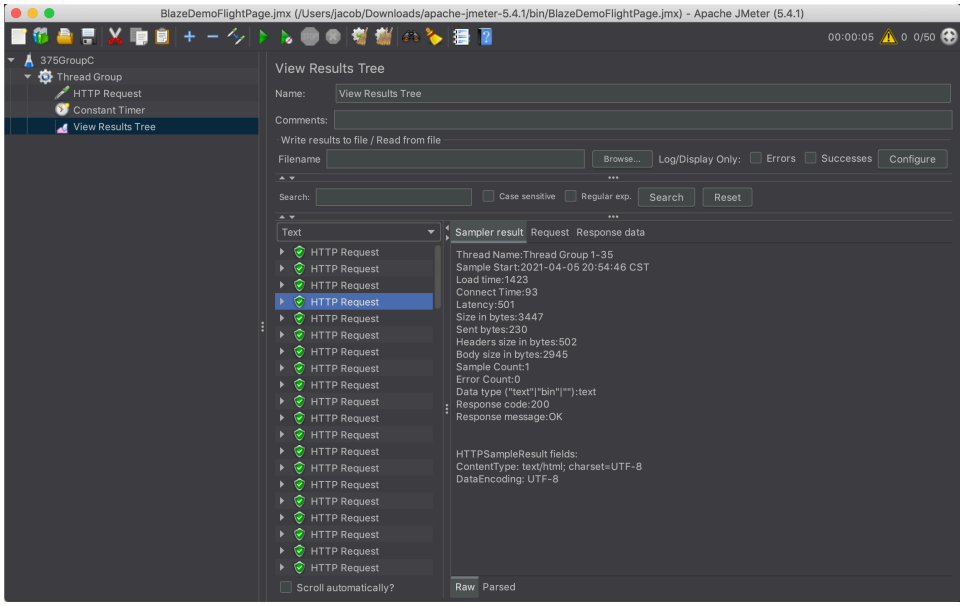
Upon scouring through StackOverflow and various GitHub repositories, we discovered that it would be necessary to download the libnss3-dev package. This was stated in multiple sources to be the only solution. However, the error still persisted even after entering the Docker container as the root and downloading the package, and as such, remains unresolved. Consequently, despite Snyk recognizing the presence of a valid version of Selenium in our POM file, we were unable to actually perform operations with Selenium in our Jenkins project.

JMeter

JMeter is a load testing tool designed to evaluate the functionality and performance of a web application when experiencing high user volume. We chose to evaluate the functionalities of JMeter with the BlazeDemo index page, which features two instances of user input; a visitor can select a departure city and a destination, and subsequently click a button to extract upcoming flights to said destination from a fake database.

As with Selenium, installing and configuring JMeter settings within Jenkins was straightforward. The steps of this process are outlined below:

1. On the Jenkins dashboard, navigate to *Configure Jenkins –> Manage Plugins.*
2. Install "Performance" from the *Available* tab.
3. Download JMeter locally, and write a basic test file with three components in the Thread Group: an HTTP Request, a 250-ms timer, and a results tree for graphical representation of the test results.



4. Run the test locally to ensure that it compiles (proof of successful execution is displayed in the screenshot on the previous page).

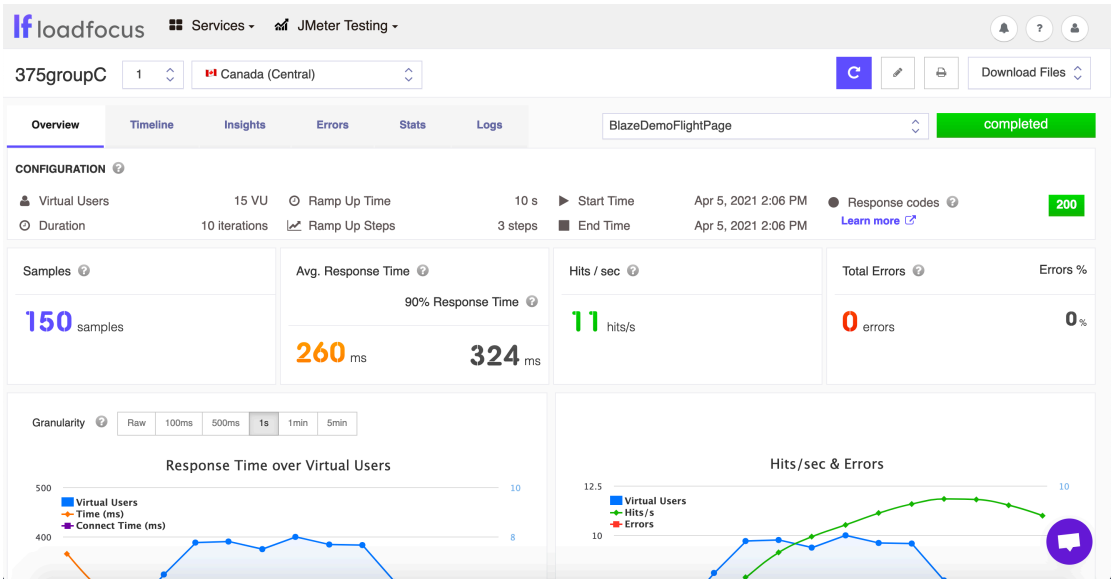
In a similar vein to Selenium, we followed multiple tutorials with different methods of invoking JMeter within the Maven project. However, we never managed to successfully execute the JMeter test in Jenkins, despite the fact that my test script worked both locally on my MacBook Pro, and remotely on LoadFocus.

The first tutorial suggested that we add a Post-Build Action to the Maven project called "Publish Performance test result." Within this window, it is possible to specify source files containing the tests; we did so, and our subsequent build failed because the file could not be found. In response, our first action was to create a file of the same name and suffix in the Jenkins slave's workspace directly. However, the build still failed with an empty-file error. Next, we uploaded the file to the GitHub repository in the hope that the Maven project would automatically detect and compile it, and in response, the build outputted the following error:

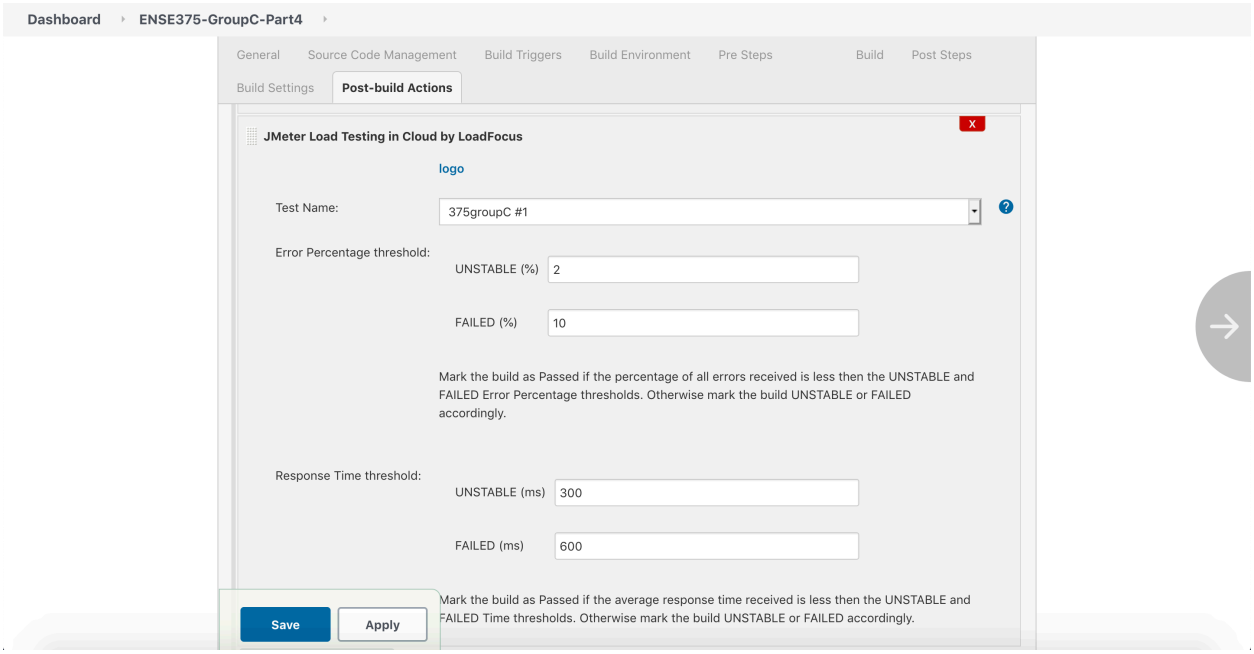
```
ERROR: Build step failed with exception
java.lang.IllegalArgumentException: Unknown xml file format
        at hudson.plugins.performance.parsers.ParserDetector.detectXMLFileType(ParserDetector.java:164)
        at hudson.plugins.performance.parsers.ParserDetector.detectXMLFileType(ParserDetector.java:138)
Caused: java.lang.IllegalStateException: XML parsing error:
        at hudson.plugins.performance.parsers.ParserDetector.detectXMLFileType(ParserDetector.java:140)
        at hudson.plugins.performance.parsers.ParserDetector.detect(ParserDetector.java:41)
        at hudson.plugins.performance.parsers.ParserFactory.getParserUsingAntPatternRelativePath(ParserFactory.java:76)
        at hudson.plugins.performance.parsers.ParserFactory.getParserWithRelativePath(ParserFactory.java:47)
        at hudson.plugins.performance.parsers.ParserFactory.getParser(ParserFactory.java:43)
        at hudson.plugins.performance.PerformancePublisher.getParsers(PerformancePublisher.java:339)
        at hudson.plugins.performance.PerformancePublisher.perform(PerformancePublisher.java:390)
        at jenkins.tasks.SimpleBuildStep.perform(SimpleBuildStep.java:123)
        at hudson.tasks.BuildStepCompatibilityLayer.perform(BuildStepCompatibilityLayer.java:80)
        at hudson.tasks.BuildStepMonitor$1.perform(BuildStepMonitor.java:20)
        at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:804)
        at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:753)
        at hudson.maven.MavenModuleSetBuild$MavenModuleSetBuildExecution.post2(MavenModuleSetBuild.java:1072)
        at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:698)
        at hudson.model.Run.execute(Run.java:1932)
        at hudson.maven.MavenModuleSetBuild.run(MavenModuleSetBuild.java:543)
        at hudson.model.ResourceController.execute(ResourceController.java:97)
        at hudson.model.Executor.run(Executor.java:429)
Build step 'Publish Performance test result report' marked build as failure
```

None of us were able to solve this error, so we attempted a second method of integrating JMeter with our project: a post-build action called "JMeter Load Testing in Cloud by LoadFocus."

Similar to Snyk, LoadFocus is a service that enables execution of system tests at multiple levels of abstraction above the source code. The user can specify a desired amount of virtual users, a duration (either in time or iterations), ramp-up time, and ramp-up steps, and upload any relevant JMeter scripts that should accompany the test. Upon completing these tasks and entering an appropriate name, the user can trigger a new test and observe the results in real time. The output of such a test on the BlazeDemo website is displayed on the next page:



The settings for the LoadFocus post-build action in our Maven project are outlined below. Note that, in order to access a test previously executed on LoadFocus, one must obtain the API key from their LoadFocus account and enter it into Jenkins' Credentials module. In addition, the screenshot displays four editable parameters that dictate the ultimate status of each build. For our project, we specified that any builds in which more than 2% of the samples yielded errors be deemed unstable, and that any builds in which more than 10% of the sample yielded errors be deemed failures. Similarly, response times of 300 and 500 ms were respectively labelled as unstable and failing conditions.



Unfortunately, for each build in which the LoadFocus test was specified as a post-build action, we encountered the following error:

```
loadfocus.com: Test Started: 375groupC
loadfocus.com: Test Config: Build UNSTABLE if errors percentage greater than or equal to 2%
loadfocus.com: Test Config: Build FAILURE if errors percentage greater than or equal to 10%
loadfocus.com: Test Config: Build UNSTABLE if response time greater than or equal to 300ms
loadfocus.com: Test Config: Build FAILURE if response time greater than or equal to 600ms
ERROR: Build step failed with exception
net.sf.json.JSONException: Invalid JSON String
        at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:143)
        at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:103)
        at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:84)
        at com.loadfocus.jenkins.api.LoadAPI.runTest(LoadAPI.java:196)
        at com.loadfocus.jenkins.LoadPublisher.perform(LoadPublisher.java:78)
        at hudson.tasks.BuildStepMonitor$3.perform(BuildStepMonitor.java:45)
        at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:804)
        at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:753)
        at hudson.maven.MavenModuleSetBuild$MavenModuleSetBuildExecution.post2(MavenModuleSetBuild.java:1072)
        at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:698)
        at hudson.model.Run.execute(Run.java:1932)
        at hudson.maven.MavenModuleSetBuild.run(MavenModuleSetBuild.java:543)
        at hudson.model.ResourceController.execute(ResourceController.java:97)
        at hudson.model.Executor.run(Executor.java:429)
Build step 'JMeter Load Testing in Cloud by LoadFocus' marked build as failure
Finished: FAILURE
```

We spent multiple days attempting to debug this error and examining documentation from StackOverflow and GitHub, but each resource that we found referenced the error in a context in which JSON files were explicitly used. Since neither our JMeter script nor our LoadFocus test contained operations for writing to or reading from a JSON file, we were unable to diagnose the issue of this error message. Therefore, despite our JMeter script succeeding locally, and our LoadFocus test succeeding within the site itself, we were unable to integrate either service into Jenkins.