The Testing Professionals

Apollo Deployment Guide

Version 0.6

This is a confidential document and is for restricted circulation only to the members of TestPro Pty Ltd

All other circulations are only on request subject to approval by TestPro.

Document Control

Version Control

Version	Date Released	Author	Remarks
0.1	19/02/2020	Dixie Fu	Initial draft
0.2	20/2/2020	Dixie Fu	Additional information included
0.3	24/2/2020	Dixie Fu	Updated after review
0.4	27/2/2020	Dixie Fu Barry Lanigan	Document re-formatted.
0.5	04/03/2020	Dixie Fu	Added detail steps and explanations
0.6	24/06/2020	Dixie Fu	Added deploying into Azure subscription through Azure Marketplace

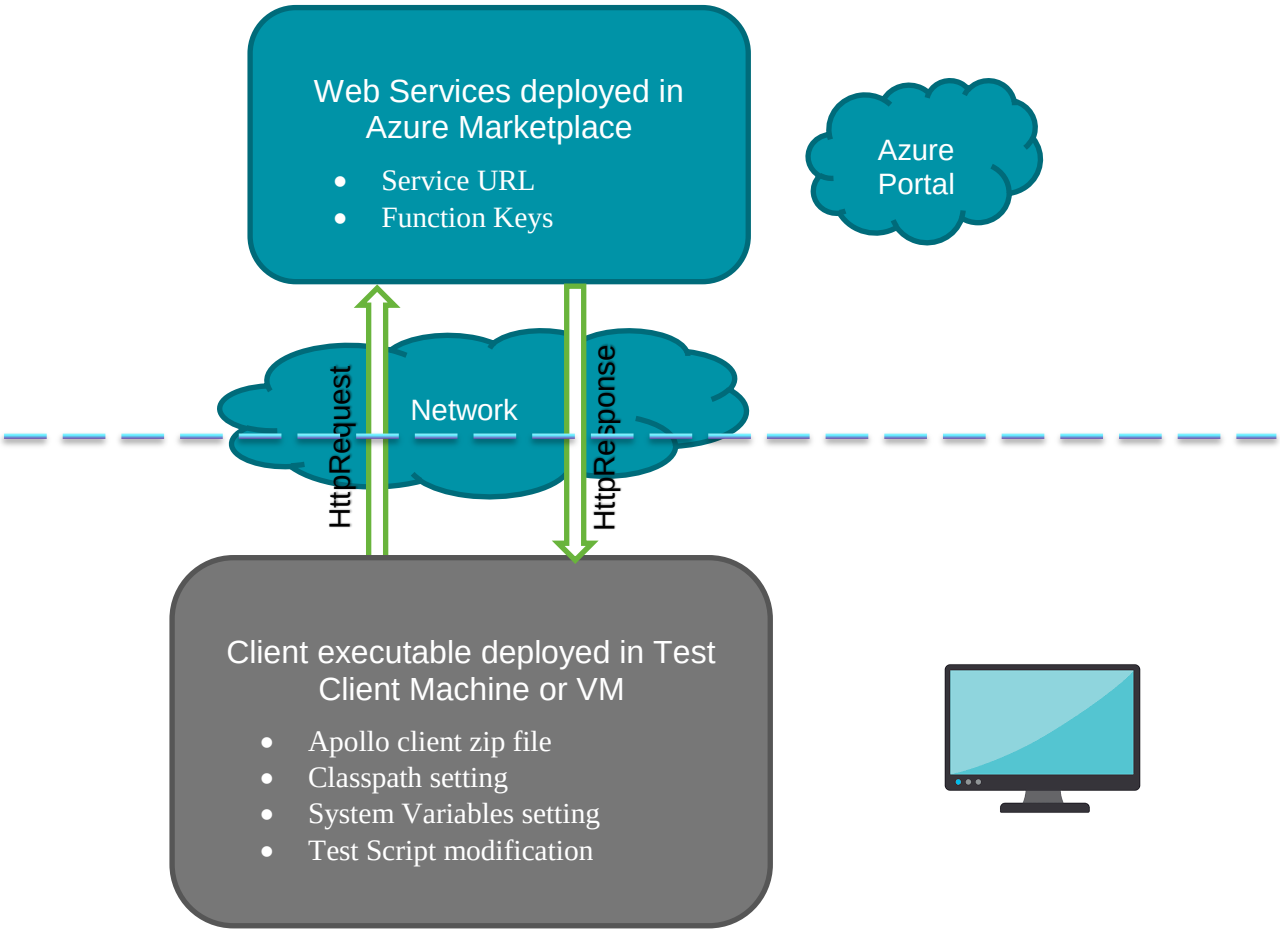
Table of Contents

Contents

1. Overview

TestPro Apollo solution consists of two parts:

1. Web Services offered in Azure Marketplace
2. Client jar deployed in Test Machine to connect to above web services



2. Deploy TestPro Apollo services through Azure Marketplace

TestPro Apollo Services are developed as Azure Functions. Customers deploy the services through Azure marketplace .

2.1 Locate TestPro Apollo offer

Go to Url: https://azuremarketplace.microsoft.com/

Search “Apollo”

Select “Apollo IBM RFT…” as illustrated below:

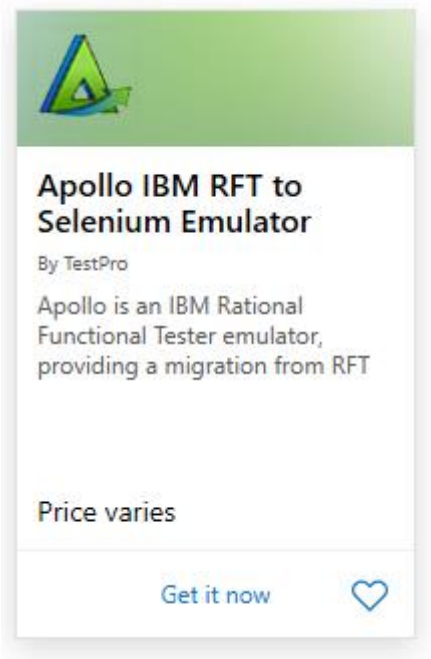


Figure 1 TestPro Apollo services

2.2 Deploy Apollo services

Click on **Get it now** in Figure 1.

Follow the prompt to select Subscription, Resource group and Region as illustrated below:

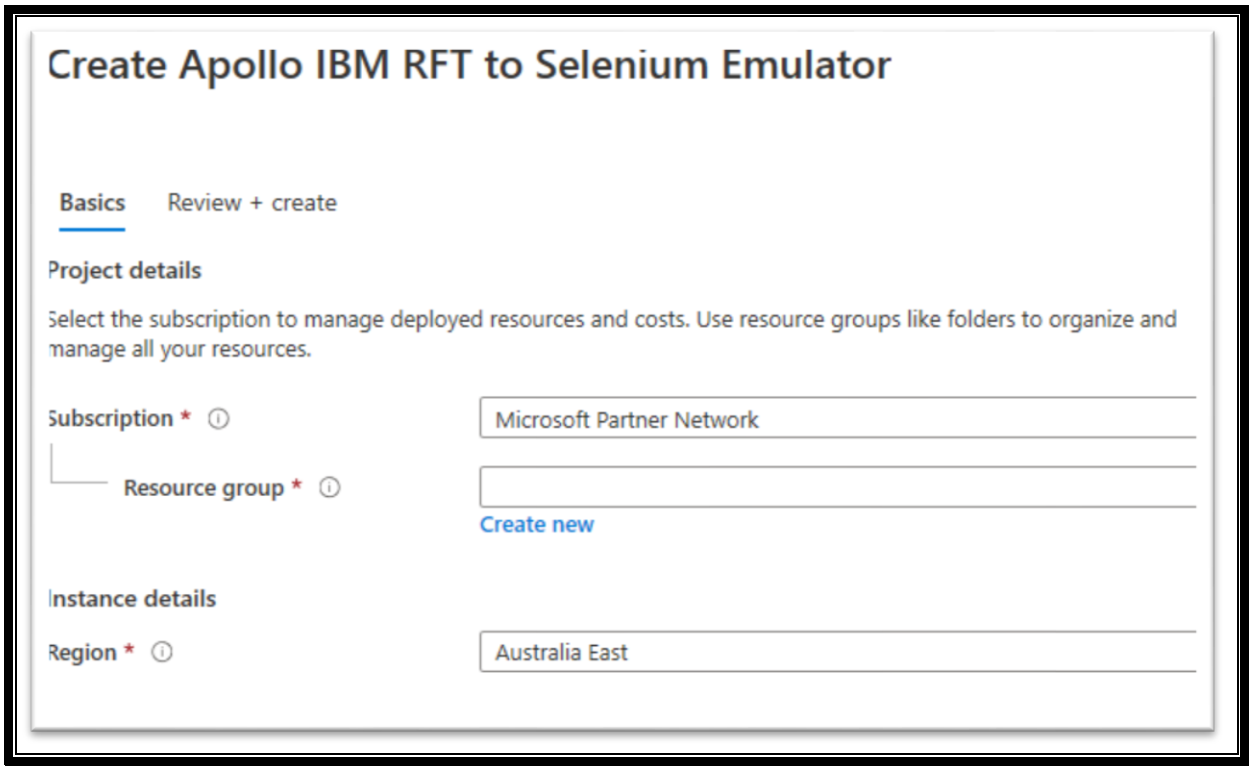


Figure 2 Subscription details

Click on **Review + create** in figure 2.

Below UI is displayed, wait until the deployment is completed.

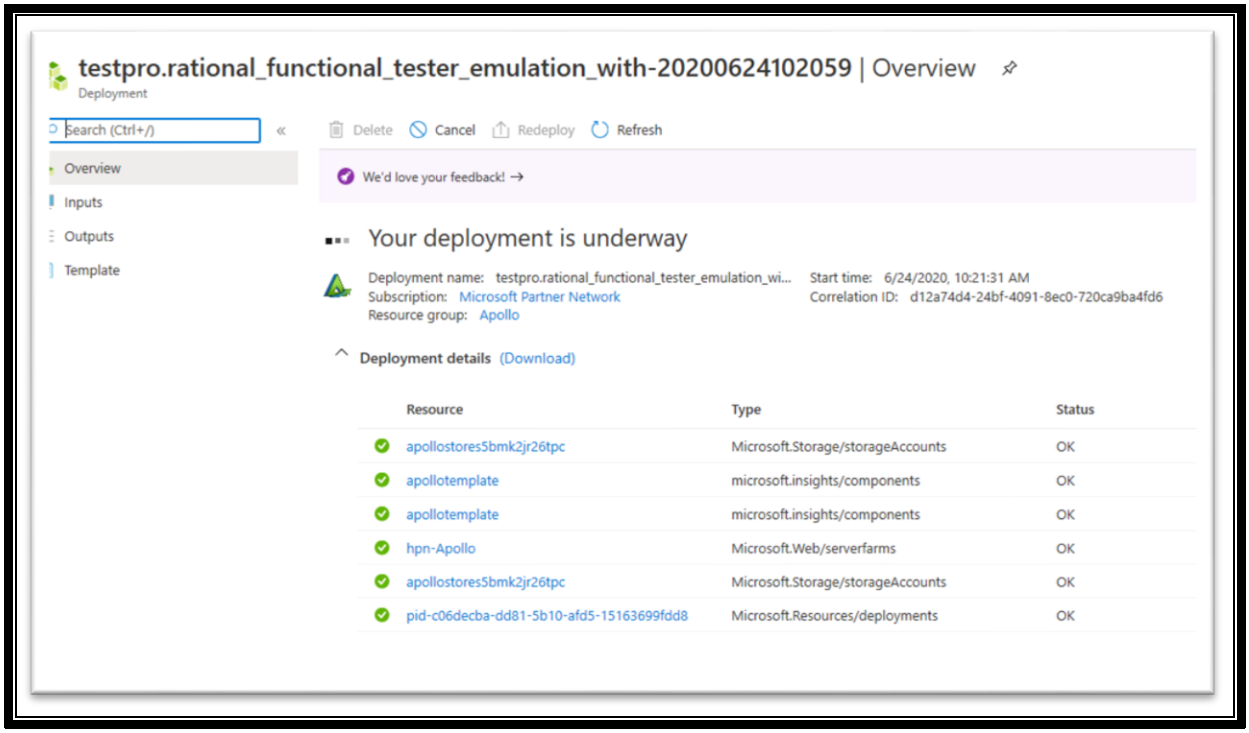


Figure 3 deploying Apollo services

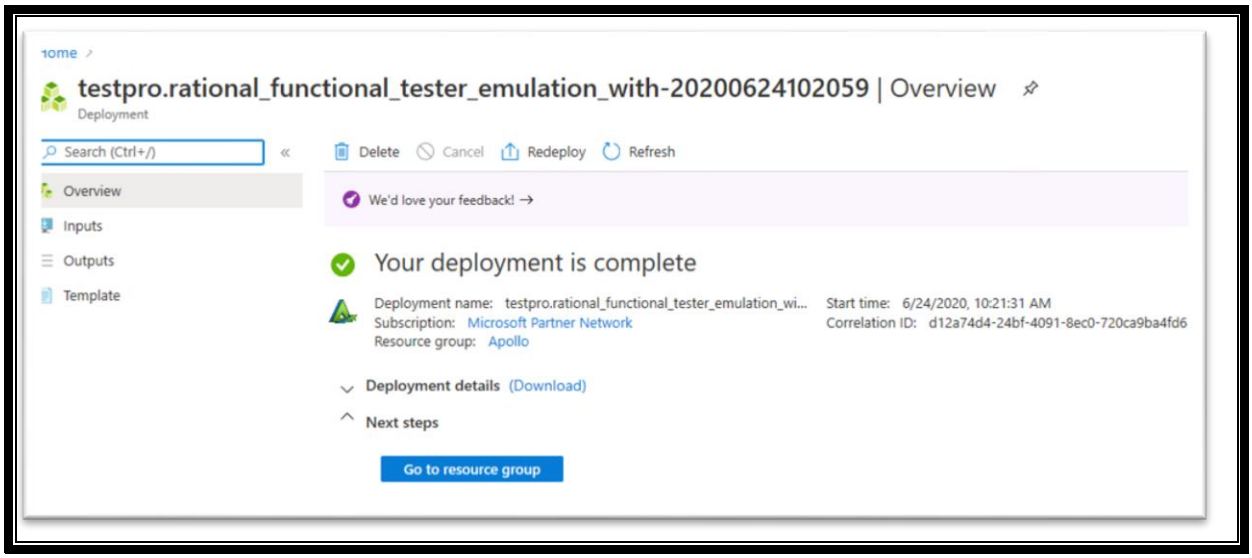


Figure 4 deployment completed

2.3 Obtain two Function keys

Click on **Go to resource group** in figure 4, below UI is shown:

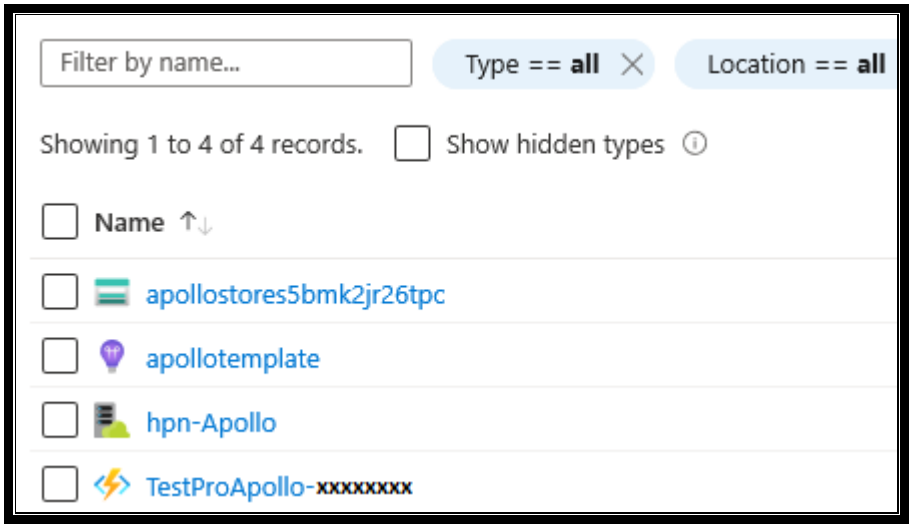


Figure 5 Apollo Azure Function

In figure 5 **TestProApollo-xxxxxx** is the Apollo services, xxxxxxx is the resource group unique id. Clicking on it brings below UI:

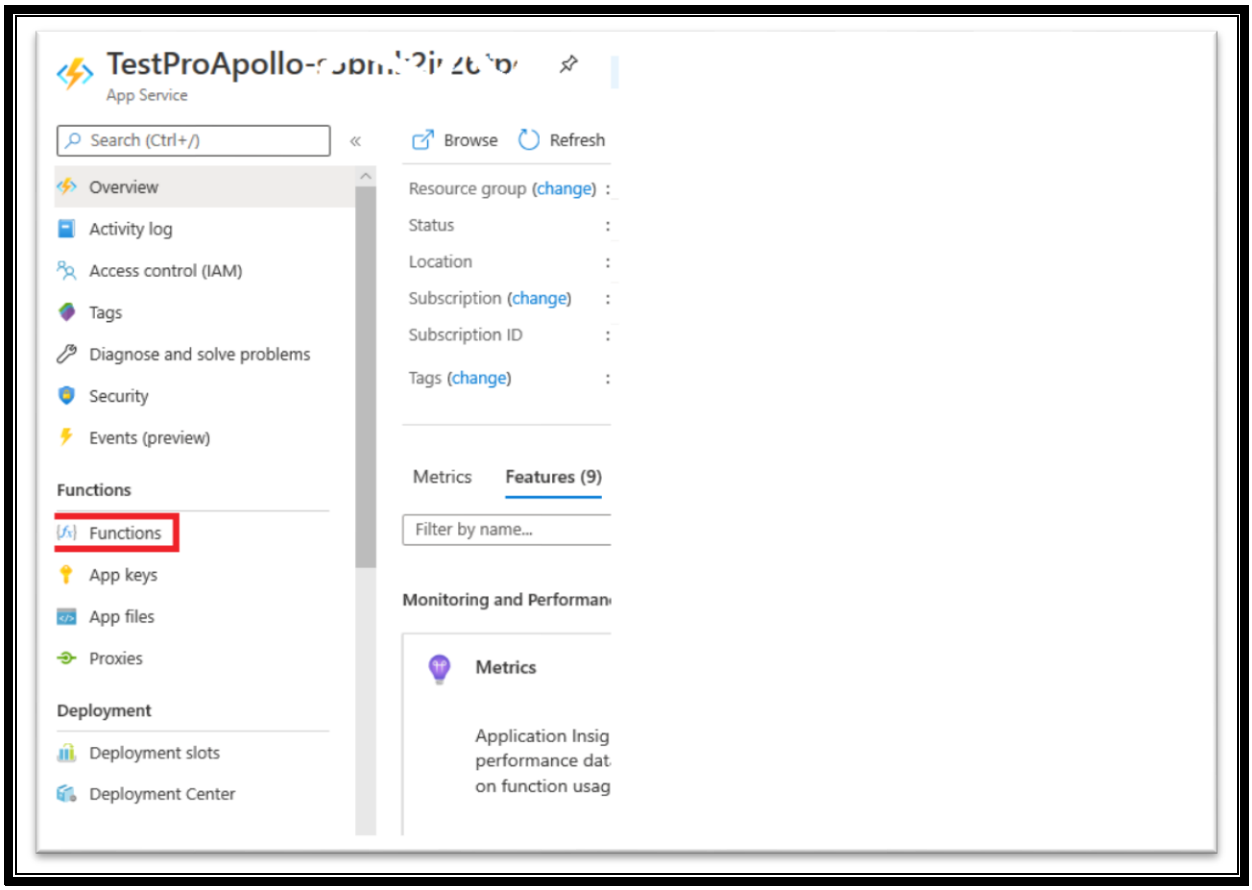


Figure 6 Functions

Click on **Functions** in figure 6 to bring below UI:

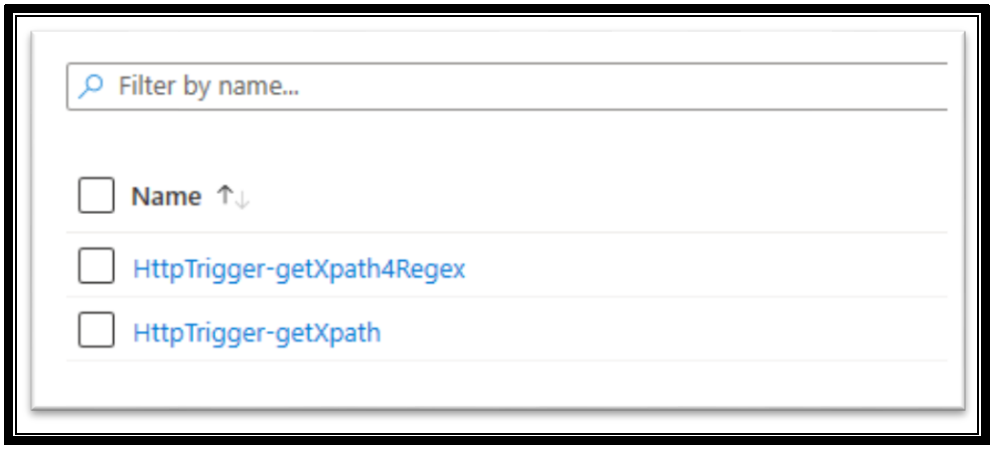


Figure 7 Function Services

,

These are the two services developed by TestPro to translate RFT find methods into Selenium calls. Click on both of them to obtain **Function Keys** to be setup as System Variables in client machines.

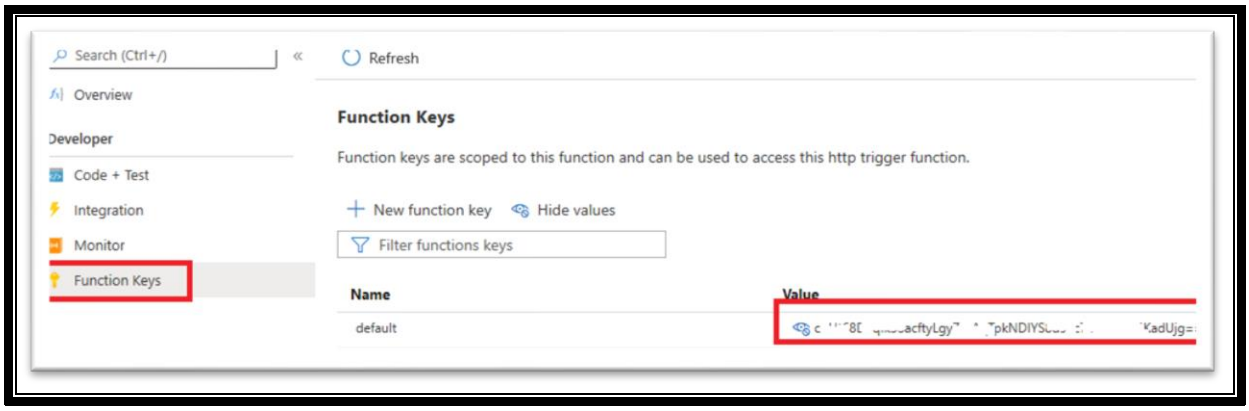


Figure 8 Function Keys

Please note that the text selection of the function key value is disabled in this Azure portal page. In order to copy out this key, user needs to right click on the value > Inspect element, then copy the value from the page source.

Put function key of ***getTrigger-getXpath*** into the test client machine's System Variable FUNCTION_KEY as described in 3.3

Put function key of ***getTrigger-getXpath4Regex*** into the test client machine's System Variable FUNCTION_RE_KEY as described in 3.3

2.4 Obtain Web Service connection details

Follow below steps to obtain Azure Function connection details after the deployment is completed:

1. login to Azure Portal → All Services → Function App
2. Obtain Service URL

   on left panel click on Function App TestPro***Apollo-<your resource group id>***, as illustrated in figure 9

Figure 9 obtain Service URL

On the right panel, obtain the service URL in format of https://testproapollo-<your resource group id>.azurewebsites.net

Put this service URL value into the test client machine's System Variable RFT2SELENIUM_EMULATOR_URL as described in 3.3

3. Test Client Configuration

3.1 Client Machine Classpath

3.1.1 Selenium WebDriver classpath

The Project Build Path needs to include below jars in order to work with Selenium WebDriver.

Unzip the selenium-java-3.141.59 .zip file to find following:

- Client-combined-3.141.59.jar (Selenium java client, version is varied)
- Commons-exec-1.3.jar
- Okhttp-3.11.0.jar
- Okio-1.14.0.jar
- guava-25.0-jre.jar
- byte-buddy-1.8.15.jar (optional, depending what selenium functions will be used by the project. TestPro Apollo Emulator doesn't need this jar)

Please note that since customers have their own preferences over Selenium Java Client version, the Apollo client distribution zip file does not include the Selenium Java Client jar files. Customers should download their own preferred version of the Selenium Java Client from www.selenium.dev/downaloads.

3.1.2 Apollo client classpath

The Project Build Path needs to include the jar files listed below in order to work with Apollo Services.

The Apollo Client Distribution zip file includes following jar files. Unzip the Apollo.zip file to find:

- httpclient-4.5.10.jar (originally obtained from Apache httpcomponents-client-4.5.10-bin.zip)
- httpcore-4.4.12.jar (originally obtained from Apache httpcomponents-client-4.5.10-bin.zip)
- commons-logging-1.2.jar (originally obtained from Apache commons-logging-1.2-bin.zip)
- commons-io-2.6 (originally obtained from Apache commons-io-2.6-bin.zip)
- Testpro-Apollo-v1.0-Selenium_3.141.59-B19.Jar (TestPro Apollo Client Jar).

  Apollo client jar file Testpro-Apollo-v1.0-Selenium_3.141.59-B19.Jar is compiled with selenium Java client 3.141.59.

3.2 Script modification

3.2.1 Update script import statements

Delete all import statements of Rational Functional Tester packages such as:

```
//import com.rational.test.ft.RationalTestException;
//import com.rational.test.ft.object.interfaces.*;
//import com.rational.test.ft.vp.PTestDataElementList;
//import com.rational.test.ft.vp.ItestDataList;
//import com.rational.test.ft.vp.ITestDataTable;
```

Import statements of TestPro Apollo client packages such as:

```
import au.com.testpro.rft2selenium.itestdata.ITestDataTable;
import au.com.testpro.rft2selenium.objects.GuiTestObject;
import au.com.testpro.rft2selenium.objects.TestObject;
import au.com.testpro.rft2selenium.objects.TextGuiTestObject;
import au.com.testpro.rft2selenium.SelectGuiSubitemTestObjects;
import au.com.testpro.rft2selenium.util.EmulatorLogger;
```

3.2.2 Extends superclass JavaSeleniumSuperClass

Every script must extend the **JavaSeleniumSuperClass** to allow RFT methods to be executed in Java run time with the Apollo client jar and Selenium libraries.

JavaSeleniumSuperClass internally defines and uses a WebDriver variable ***driver*** to translate RFT TestObject operations into Selenium Element operations. Every script must instantiate variable ***driver*** otherwise NullPointerException will be thrown. Please note that scripts cannot re-declare variable WebDriver ***driver*** again i.e. scripts are expected to strictly use variable ***driver*** which is defined in **JavaSeleniumSuperClass** and instantiate it.

For convenience the JavaSeleniumSuperClass defines a static method startBrowser(String browser, String URL) to assign basic WebDriver types to ***driver*** and launch the web page defined by parameter URL, as illustrated in *Figure 1*

Scripts can call startBrowser(browser, URL) directly to instantiate variable ***driver*** and launch the web page.

Scripts can instantiate ***driver*** in their own specific way. For instance, instantiate a ChromeDriver using ChromeOptions as constructor parameter.

```
public void startBrowser(String browser, String URL) {

        homeURL = URL;

        if (browser.equalsIgnoreCase("FireFox")) {
              driver = new FirefoxDriver();
        } else if (browser.equalsIgnoreCase("Chrome")) {
              driver = new ChromeDriver();
        } else if (browser.equalsIgnoreCase("Edge")) {
```

```
            driver = new EdgeDriver();
        } else { // default
            driver = new InternetExplorerDriver();
        }

        driver.get(URL);
    }
```

Figure 10: driver initialization in JavaSeleniumSuperClass

3.3 Create System Variables

Create System Variables listed as below.

The actual values will become available upon the completion of Apollo Marketplace deployment as described in section 2.3

System Variable Name	Example Value
RFT2SELENIUM_EMULATOR_URL	https://apollo-sxxxxxxxxxxxxxxx.azurewebsites.net/
FUNCTION_KEY	v01S8QIGbXlnagdUm93345uuXzgmJ123456463VVduFBZ1CWgk;lllkkpQ==
FUNCTION_RE_KEY	RWHEtdYKwd3IulW8Am5iD6phuu9R5KVU7VnOUQSAxnKtX123456yI5Q==

4. APOLLO Client Scripting Guide

4.1 Logging

Once the test script extends **JavaSeleniumSuperClass**, the script can use variable **logger** straight away without having to instantiate it.

- available methods for variable **logger**
  - **logger**.logMessage(String message)
  - **logger**.logInfo(String information)
  - **logger**.logError(String errorMessage)
  - **logger**.logError(String errorMessage, Throwable t): print the error message along with the Throwable's stacktrace to the log
  - **logger**.logWarning(String warningMessage)
  - **logger**.logException(Throwable t): print the Throwable's stacktrace to the log
  - **logger**.getLogPath()
- The log file is located at: Test-Script-Project-Path/Logs/Emulator.log.yyyy-mm-dd
- One new log file will be created each day
- If the project is in a shared drive and multiple test machines are pointing to the same project then all logging will end up in the same log file
- logMessage(), logInfo(), logWarning(), logError() produce logging as below format:

```
21 Feb 2020 16:44:13 INFO aaaaaaaaaaaaaaaaaa
21 Feb 2020 16:44:13 INFO bbbbbbbbbbbbbbbbbb
21 Feb 2020 16:44:13 WARN cccccccccccccccccccccc
21 Feb 2020 16:44:13 ERROR ddddddddddddddddddddd
```

Use below statement to obtain the **logger** handler if a Java class doesn't extend JavaSeleniumSuperClass:

```
Private static EmulatorLogger logger =
EmulatorLogger.getLogger(JavaFrameworkSuperClass.getLogPath());
```

4.2 Log execution result

Calling setPass(), setWarn() and setFail() in a script to log execution result.

Method call	Message printed to the logger
setPass()	Execution of Script "xxxxx" is passed
setWarn()	Execution of Script "xxxxx" has warning
setFail()	Execution of Script "xxxxx" is failed