Splunk App for Synthetic Transaction Monitoring

Presented by: Elias Haddad

Contents

Release Notes 3

What’s New 3

Current known issues 3

Introduction 3

Architecture 3

Configuration on the Splunk Forwarder 4

Testing your configuration. 5

Recording Selenium Scripts 5

Defining Transactions in the Selenium Script 6

Running the Selenium Script in Splunk 7

# Release Notes

## What’s New

This app has been tested on Mac, Liniux and Windows.

This app has been tested with Safari, Firefox and Chrome. IE is supported but not yet tested.

## Current known issues

Selenium Webdriver has an issue with Safari version 7.1 o Mac OS X. Please check troubleshooting section for more details

# Introduction

The Synthetic Monitoring App for Splunk enables you to monitor your Web application and measure critical KPI metrics such as application performance and availability. Built on the Selenium WebDriver, this app allows you to proactively detect application problems before your customers do. With the synthetic monitoring application, you will be able to simulate user interactions around the clock and set up alerts when your application breaches its performance and availability SLAs. This app allows you to compare the end user performance of your web application from different locations, various browsers and from a myriad of devices.

Selenium Scripts can be recorded using the Selenium IDE, which is a Firefox add-on that allows you to record any browser interactions and export these interactions in Python, Java or Ruby scripts.

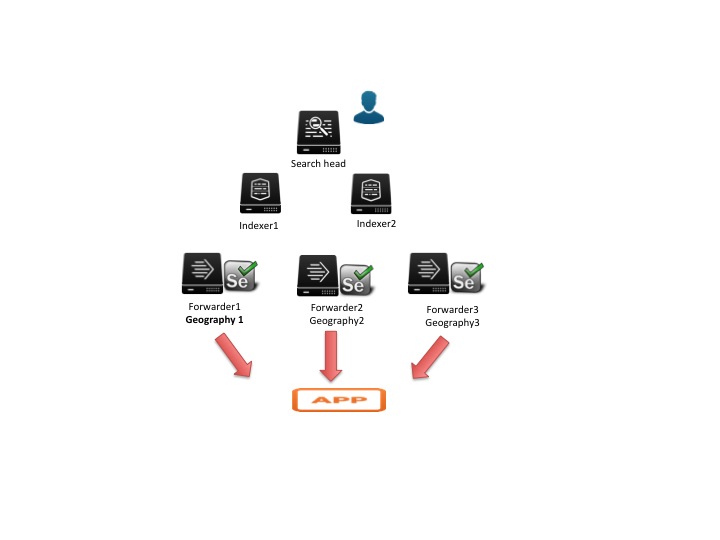
Since Selenium is an automation tool, it does not have a built-in mechanism to measure response time. In order to record the transaction response time, the app comes with a python module “Splunk Transaction” that allows you to define transactions and measure various metrics and index the results in Splunk.

The App also allows you to measure latency between the Splunk forwarder and the Application that is being monitored. By comparing End User response time measured by the Selenium Script and the network latency, you will be able to isolate network latency from application performance problems.

# Architecture

The app sould be installed on the Splunk Universal forwarder in order to replay the Selenium scripts as well as on the search head for the visualization layer. Ideally the Splunk Forwarder needs to exist in the location of your highest concentration of users in order to better represent the performance of the users in that location. To better manage the distributed nature of the app, a Splunk deployment server can be used in order to manage the distributed nature of the app.

In order to replay the Selenium Scripts, the app requires Python to be installed. It is also required to import the UserAgents and Selenium Modules (for more details please refer to the configuration section, below). It is recommended not to import these modules in the Splunk Python but install a separate python and import the modules there to avoid tampering with the Splunk built in Python. Since Selenium requires the browser used in the simulation to be installed on the same server, you will need to install the browser involved with the simulation on all the forwarders where the scripts need to be executed.



# Configuration on the Splunk Forwarder

1- Install the app on the Splunk forwarder by extracting the tar file or doing it from the UI

2- Install Python 3.4 or above outside of the Splunk directory path on the same Splunk forwarder

3- Install selenium and user-agents modules using pip. To do so, you can follow the steps below:

a- From the newly installed python directory , go to the bin folder and run the following commands:

b- pip3.4 install user\_agents

c- pip3.4 install selenium

For more details on how to configure python for selenium, refer to the following

<http://selenium-python.readthedocs.org/en/latest/installation.html>

4- Download the Selenium WebDriver and place it somewhere on the same splunk machine.

<http://selenium-release.storage.googleapis.com/index.html?path=2.43/>

5- If you want to simulate Chrome browser interactions, download the latest Chrome Driver from:

<http://chromedriver.storage.googleapis.com/index.html> and place the downloaded artifact under /usr/bin on the machine where you will be replaying your scripts.

6- Since Selenium requires the actual browser to be installed in order to replay the scripts; you will need to install the necessary browser(s) on the forwarder machine. The supported browsers are Opera, Firefox, Safari and IE.

# Testing your configuration.

In order to test your configuration, the app is bundled with sample Selenium scripts with Splunk Transactions. You will need to enable the corresponding scripted inputs from Settings-> Data Inputs -> Scripts in the UI.

The app is also bundled with a sample ping monitor that allows you to monitor the network latency from your Splunk Forwarder to [www.python.org](http://www.python.org) server.

Once you have enabled the scripts you should be able to start seeing data indexed. You can confirm the same by running the following search:   
 > index=synthetic.

# Recording Selenium Scripts

1- Download and install the Selenium IDE on your Firefox browser. This will allow you to record Selenium scripts and export the interaction in a Python script:

<https://addons.mozilla.org/en-US/firefox/addon/selenium-expert-selenium-ide/>

2- Make sure that the script replays successfully in the Selenium IDE.

3- Once the script is successfully recorded, export the script from the Selenium IDE menu

(File -> Export Test case as -> Python/Junit/Webdriver). Make sure the script is saved with .py extension

# Defining Transactions in the Selenium Script

1- Edit the newly created python Selenium script to add the splunk transaction definitions as follow:

###### STEP 1

### Include the splunktransactions module in your script

from splunktransactions import Transaction

###### STEP 2

### Provide a name to your Application that this script is trying to monitor eg. 'Google'

a=Transaction(driver, 'Google')

###### STEP 3

### assign a name to the transaction by defining a start and an end to any part of the script

a.TransactionStart(driver, 'Google Home Page')

driver.get(self.base\_url + "/")

a.TransactionEnd(driver, 'Google Home Page')

###### STEP 4

### Repeat Step 3 and 4 as needed for as many transactions as needed

2- Save the Python script on the <Splunk>/etc/apps/splunk-app-synthetic/bin folder on the server where the script will be replayed

3- Create a batch/shell file under the same <Splunk>/etc/apps/splunk-app-synthetic/bin folder that invokes the newly generated python script and sets the right Selenium server in the class path. Example shell script:

#!/bin/bash

unset PYTHONPATH

unset LD\_LIBRARY\_PATH

CLASSPATH="<change me>/selenium-server-standalone-2.40.jar"

export CLASSPATH

SELENIUM\_SERVER\_JAR="/<change me>/selenium-server-standalone-\*.jar"

export SELENIUM\_SERVER\_JAR

python3.4 <change me>/etc/apps/splunk-app-synthetic/bin/GoogleTest.py

4- Make sure that the script runs successfully from the command line

# Running the Selenium Script in Splunk

1- Create a scripted input that invokes the shell script. Define the frequency of execution in the scripted input

2- If you want a ping monitor, create another scripted input to point to the ping.py file included with the app.

Example inputs.conf entry to ping www.python.org :

[script://./bin/ping.py www.python.org]

disabled = 1

index = synthetic

interval = 60.0

source = ping

sourcetype = network

3- Repeat steps 1 and 2 as needed

4- Use the Splunk deployment server to push the scripted inputs to the Splunk forwraders as needed.

# Troubleshooting

1- validate that your environment is properly set by setting the

2- Make sure you run the script manually outside of Splunk using Python before testing using the scripted inputs