Testing GUI Apps with Sikulix & Dynatrace Synthetic

Contents

[Requirements 3](#_Toc92387869)

[ActiveGate settings 4](#_Toc92387870)

[Dynatrace bridge 4](#_Toc92387871)

[Sikulix requirements 4](#_Toc92387872)

[copy files 5](#_Toc92387873)

[Start program 5](#_Toc92387874)

[Install as a service 5](#_Toc92387875)

[Bridge features 5](#_Toc92387876)

[POST /testtool\_launcher2 6](#_Toc92387877)

[Bridge side 6](#_Toc92387878)

[Dynatrace side 10](#_Toc92387879)

[GET /testtool\_properties 12](#_Toc92387880)

[Bridge side 12](#_Toc92387881)

[Dynatrace side 13](#_Toc92387882)

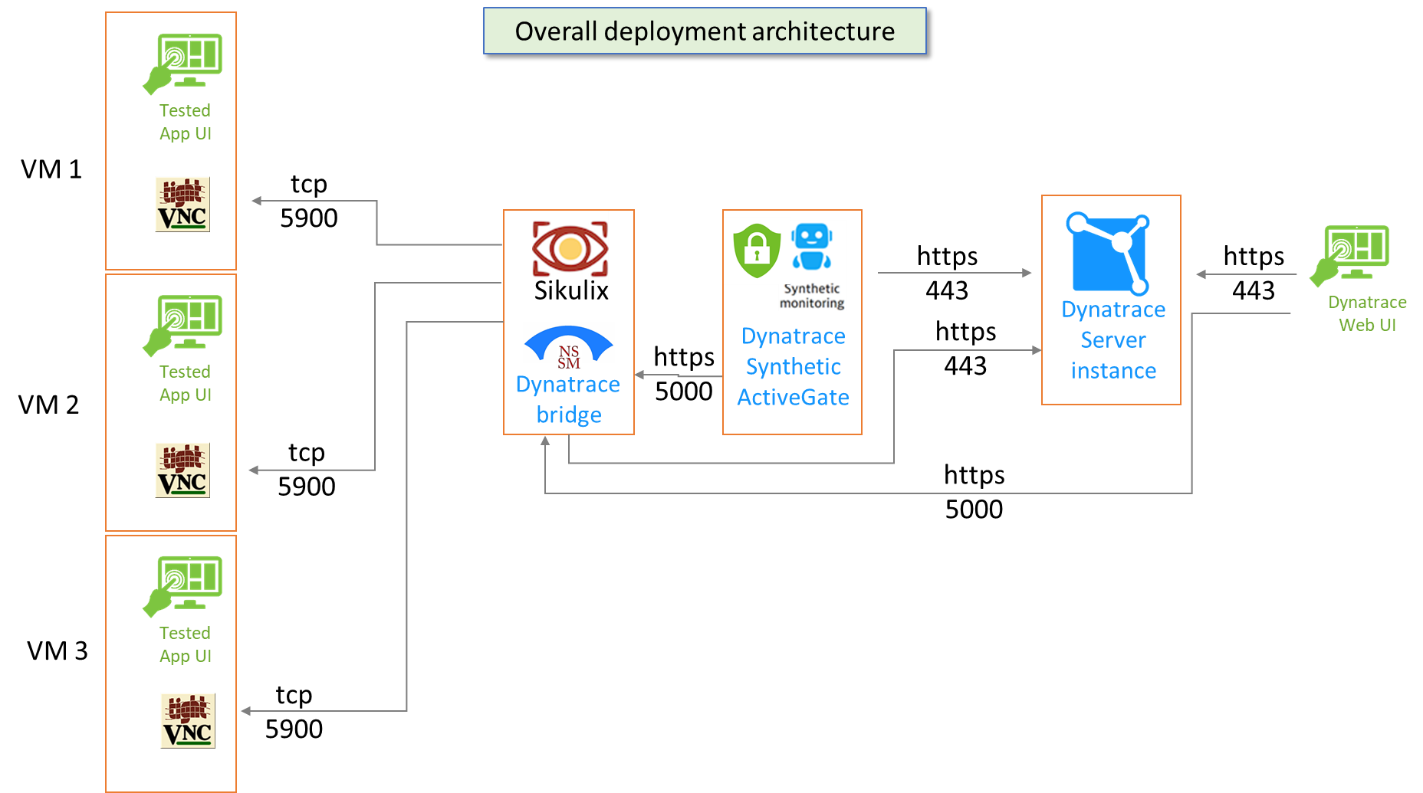
[POST /testtool\_launcher 14](#_Toc92387883)

[Bridge side 14](#_Toc92387884)

[Dynatrace side 17](#_Toc92387885)

Author : [laurent.izac@dynatrace.com](mailto:laurent.izac@dynatrace.com)

Credits : [eric.maillard@dynatrace.com](mailto:eric.maillard@dynatrace.com)



# Requirements

Testing GUI Apps with Dynatrace Synthetic requires the following components :

* Dynatrace server instance : hosts the test settings and results ; will be accessed by end users through the Web UI.
* Dynatrace Synthetic ActiveGate : runs Synthetic tests using embedded Web browser and http client.
* Dynatrace Sikulix bridge :
  + acts as a proxy between Dynatrace and Sikulix.
  + Embeds Sikulix runtime to run GUI navigation scenarii against various client Apps.
  + Provides response times, errors and screenshots through a Web UI.
  + Sends test execution results as Third Party monitor in Dynatrace.
  + Can be installed as a service when used to test remote Desktop App accessed through VNC.
* TightVNC Server :
  + Allows remote connection to the Desktop session, with ability to open and close Desktop session as part of a SikuliX script.

Notes :

* Dynatrace Server and ActiveGate are the stock Dynatrace products.
* Dynatrace Sikulix bridge is an adhoc script written in Python.
* Sikulix is an [Opensource](http://sikulix.com/quickstart/) GUI testing tool.
* TightVNC Server is a [free](https://www.tightvnc.com/licensing-tvnserver.php) (GNU GPL) VNC Server available for Windows and Linux.

## ActiveGate settings

## Dynatrace bridge

### Sikulix requirements

Sikulix requires a 64 bits Windows system, a 64 bits Java runtime (1.11+) and needs to have access to screen, mouse and keyboard to be able to execute tests.

Note : It also requires « Media Foundation » feature to be installed on the system (in case it’s not there by default) :

Graphical user interface

Description automatically generated

Note : if installed on the same Windows machine as the Dynatrace Synthetic ActiveGate, it will be able to use the already installed jre

(found in « c:\Program Files\dynatrace\synthetic\jre\bin\java.exe »)

Note : The Sikulix runtime is a Jar file (sikulixide-2.0.5.jar) bundled with the Dynatrace Sikulix bridge.

### copy files

Unzip the [Dynatrace bridge files](https://dynatrace-my.sharepoint.com/:u:/p/laurent_izac/EbtTnihxhOZBnm91KQrf6WMBFypbTc1Ai3tmDdXCKvSj_Q?e=PAbm8Z) in a target folder…

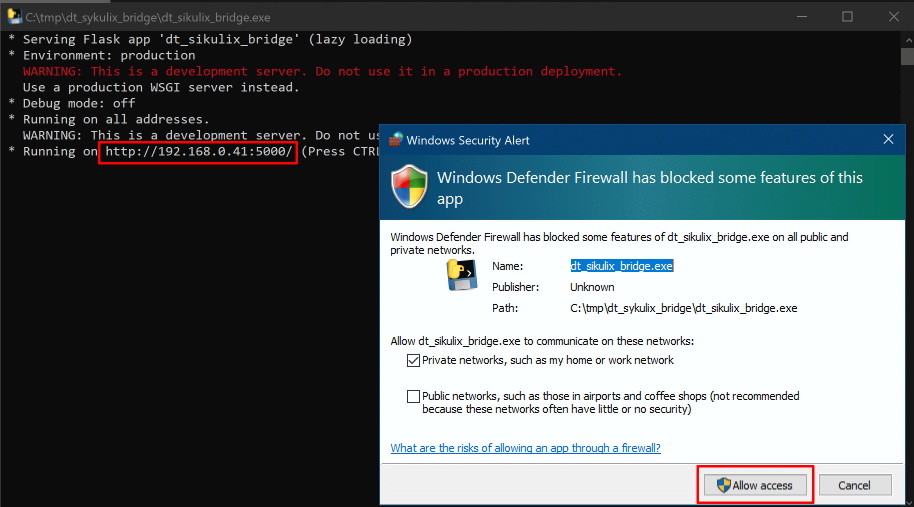
### Start program

* On Windows, From the Explorer, double click on « run\_bridge.bat » to start the bridge.
* On Linux, from the shell, execute « run\_bridge.sh » to start the bridge.

Note : by default, the batch file is trying to use the Java runtime from the Synthetic ActiveGate.

Take note of the URL the server is running on.This will be used from Dyntrace Synthetic script to trigger Sikulix execution.

On Windows, if you are presented with a security dialog box, Click on « Allow access » to let the program run.



### Install as a service

When using VNC to remotely connect to a Desktop application for UI testing, the bridge can be run without access to local screen, keyboad and mouse. Thus it can run as a background task, i.e. a service.

To install the bridge as a service :

* On Windows : From Windows Explorer, double click on « install\_dt\_automation\_bridge\_as\_a\_service.bat».
* On Linux, from the shell, execute « install\_dt\_automation\_bridge\_as\_a\_service.sh».

# Bridge features

The bridge provides :

* A web UI :
  + To manage deployed scripts (add, remove) ;
  + to view execution results (screenshots, response times, errors)
* A REST API :
  + To trigger script executions
  + To retrieve script execution context data.

## Web UI

## REST API

### POST /testtool\_launcher2

Triggers a script execution with the following parameters in a form (application/x-www-form-urlencoded) :

* + script: name of the script to run ;
  + api\_url: Dynatrace endpoint to send execution results to, through Third Party API ;
  + api\_token : Dynatrace API token with « Create and read Synthetic monitors and nodes » V1 access scope ;
  + [<param1>…<paramn>] : any number of parameters, as required by the script to execute.

Example POST body content :

A screenshot of a computer

Description automatically generated with medium confidence

#### Bridge side

Below is an example Sikulix script trying to :

* Connect to a VNC server ;
* Open a Windows session ;
* Start the calculator app ;
* Make some calculations ;
* Close the app ;
* Close the session ;
* Disconnect from the VNC server.

# import the Dynatrace Bridge features

load("dtbridge\_sikuli.jar")

from dtbridge import \*

#--------------------------------------------------------------------------------

@dt\_get\_timings

def open\_session():

    if(exists("PressCtrlAlt.png")):

        dt\_vnc\_send\_ctrl\_alt\_del()

        wait(2)

    if(exists("password\_field.png")):

        type("replace with your password")

        type(Key.ENTER)

        wait(2)

#--------------------------------------------------------------------------------

@dt\_get\_timings

def close\_session():

    # lock the session when finished

    dt\_vnc\_send\_ctrl\_alt\_del()

    click(wait("Signout-1.png",5))

#--------------------------------------------------------------------------------

@dt\_get\_timings

def open\_calc():

    # minimize all open windows

    type("M", Key.WIN)

    wait(5)

    # start calculator App

    click("windows\_logo.png")

    wait(5)

    type("calc")

    wait(2)

    type(Key.ENTER)

    # wait for the app to start

    wait("flCalcul.png",5)

#--------------------------------------------------------------------------------

@dt\_get\_timings

def calc1():

    # interact with the mouse

    click("eight\_digit.png")

    click("multiply.png")

    click("seven\_digit.png")

    click("equal\_sign.png")

    wait(1)

    # check result

    wait("expected\_56.png",1)

#--------------------------------------------------------------------------------

@dt\_get\_timings

def calc2(entry):

    # interact with key strokes

    type(entry+Key.ENTER)

    # check result

    wait("expected\_1dot2.png",1)

#--------------------------------------------------------------------------------

@dt\_get\_timings

def close\_calc():

    # close the app

    click("window\_close.png")

#--------------------------------------------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

   # build a list of passed parameters

   args=dt\_get\_args(sys.argv)

   vs=dt\_vnc\_connect("replace with you VNC server address")

   # make it the default screen for the following commands

   use(vs)

   open\_session()

   try:

       open\_calc()

       calc1()

       if 'param1' in args:

           calc2(args['param1'])

       else:

           calc2("6/5")

       close\_calc()

   finally:

       close\_session()

       dt\_vnc\_disconnect()

       # revert to default local screen

       use()

Here is what the script looks like when opened from the Sikulix IDE :

Graphical user interface, text, application

Description automatically generated Graphical user interface, text

Description automatically generated

Note : this sample script can be downloaded from the [Github repository](https://github.com/LO-RAN/dt_sikulix_bridge/tree/main/documentation/examples).

Note : the Sikulix IDE can be executed with the following command :

* On Windows, From the Explorer, double click on « run\_ide.bat ».
* On Linux, from the shell, execute « run\_ide.sh ».
  + Note: on Linux, a X window graphical interface is required.

Note : once the script has been tested in interactive mode within the IDE, it is time to compile it as a jar file (« CTRL+J » or « File > Export as jar ») for execution on the bridge.

#### Dynatrace side

Below is an example HTTP Monitor step that will trigger the Sikulix script by calling the « /testtool\_launcher2 » POST URL :

Graphical user interface, application

Description automatically generated

A post-execution srcript allows to handle the error condition and create a event/problem:

// check response code

if (response.getStatusCode() != 200) {

// if not 200, then fail

api.fail(response.getResponseBody());

}

Note: in this example, the API token has been saved in the Credential Vault so that it does not appear as clear text. The same can be done with any required sensitive parameter…

Every time the Dynatrace http Monitor Synthetic script runs, say every 15 mn, it will trigger a request to the bridge, that will aknowledge the request or return an error status.

Any bridge availability issue or parameter issue will be reported as a failure (event/problem).

The http monitor is not

Graphical user interface, application, Teams

Description automatically generated

In case of error, the Sikulix error message is available in « Analyze last execution »  :



## GET /testtool\_properties

Gets a properties file used to gather context data for a subsequent script execution.

### Bridge side

Below is an example json file containing properties that can be retrieved by calling the « /testtool\_properties » URL.

{

  "SIKULI\_SCRIPT" : "C:\\tmp\\Sikulix\\scripts\\office",

  "word\_doc\_1" : "MyTestWordDocument"

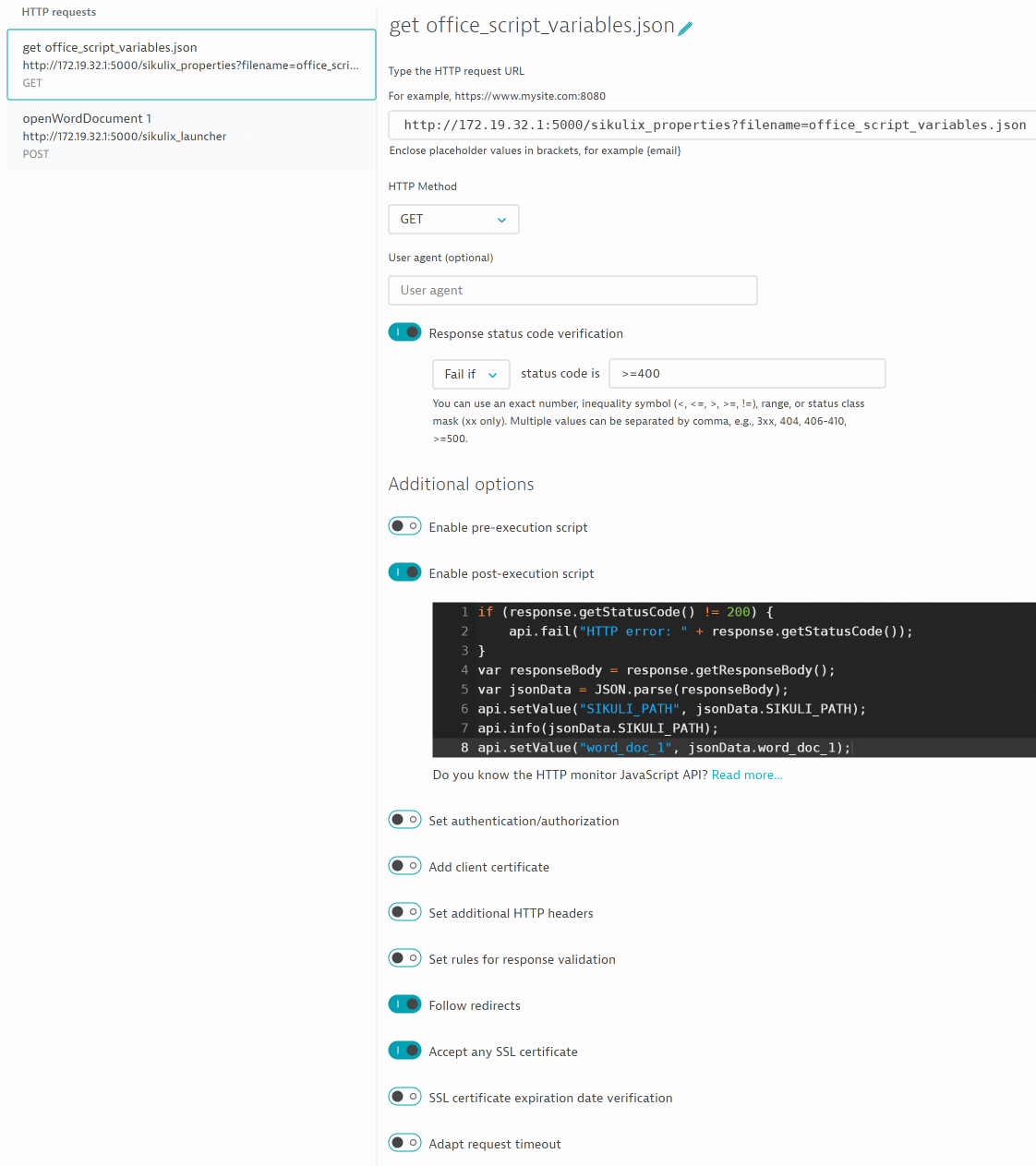
}

Note : the first key-value pair (SIKULI\_SCRIPT) is used to locate the Sikuli script to execute. All other key-value pairs will be adapted to your test context (set both keys and values according to your testing needs).

### Dynatrace side

Below is an example HTTP Monitor step that will retrieve the contents of these json properties :

A « GET » request is sent to the bridge, calling the « /sikulix\_properties » URL, with a « filename » parameter pointing to the pathname of the json file to retrieve :



A Post-execution script processes the json structure and extracts the contents to build 2 placeholders that will be used in the following steps :

if (response.getStatusCode() != 200) {

api.fail("HTTP error: " + response.getStatusCode());

}

var responseBody = response.getResponseBody();

var jsonData = JSON.parse(responseBody);

api.setValue("SIKULI\_PATH", jsonData.SIKULI\_PATH);

api.info(jsonData.SIKULI\_PATH);

api.setValue("word\_doc\_1", jsonData.word\_doc\_1);

## POST /testtool\_launcher

Triggers a script execution with the following parameters in a json structure :

* + script\_name : pathname of the script to run ;
  + function\_to\_execute : name and arguments of function to execute in the script

### Bridge side

Below is an example Sikulix script trying to open a Word document with parameters taken from a previously retrieved json properties content :

The POST body content is a json structure like the following :

{

  "script\_name" : " C:\\tmp\\Sikulix\\scripts\\office.sikuli",

  "function\_to\_execute" : "openWordDocument('MyTestWordDocument')",

}

From this content, the bridge calls the following « office.sikuli » script :

import sys

def openWordDocument(documentName):

click("1625821325583.png")

wait(1)

type("word")

click("1625821499357.png")

wait(1)

click("1625821549524.png")

click("1625821587111.png")

paste('C:\\my test folder\\my word documents')

click("1625821764224.png")

if (documentName == MyTestWordDocument'):

click("1625821716905.png")

elif (documentName == ' MyTestWordDocument 2'):

click("1625822469097.png")

elif (documentName == ' MyTestWordDocument 3'):

click("1625822510497.png")

else:

print('[error] word document does not exits : '+documentName)

click("1625821764224.png")

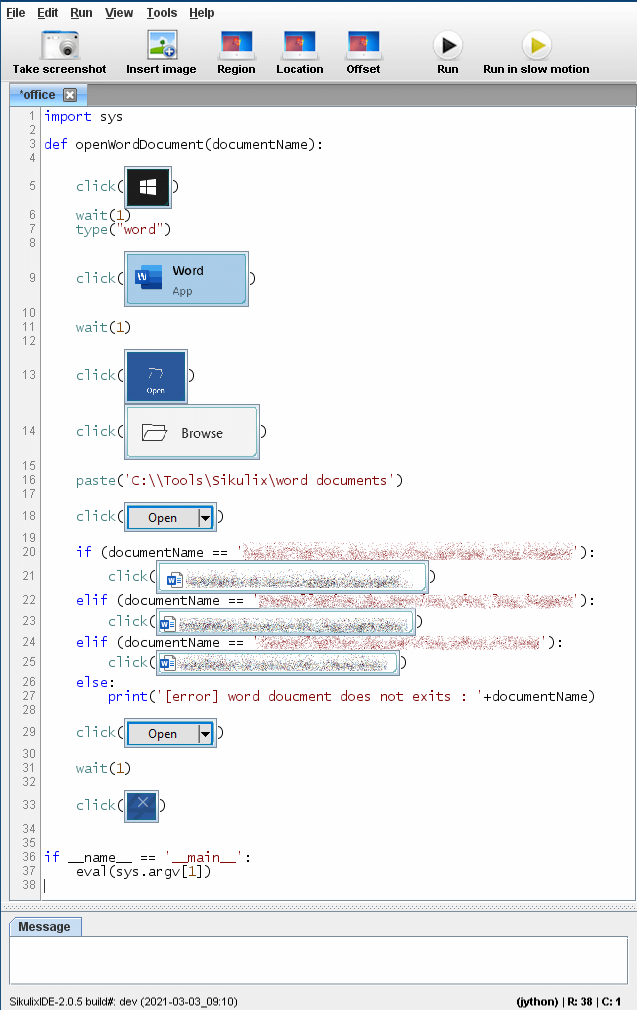
wait(1)

click("1625837585975.png")

if \_\_name\_\_ == '\_\_main\_\_':

    eval(sys.argv[1])

Here is what the script looks like when opened from the Sikulix IDE :

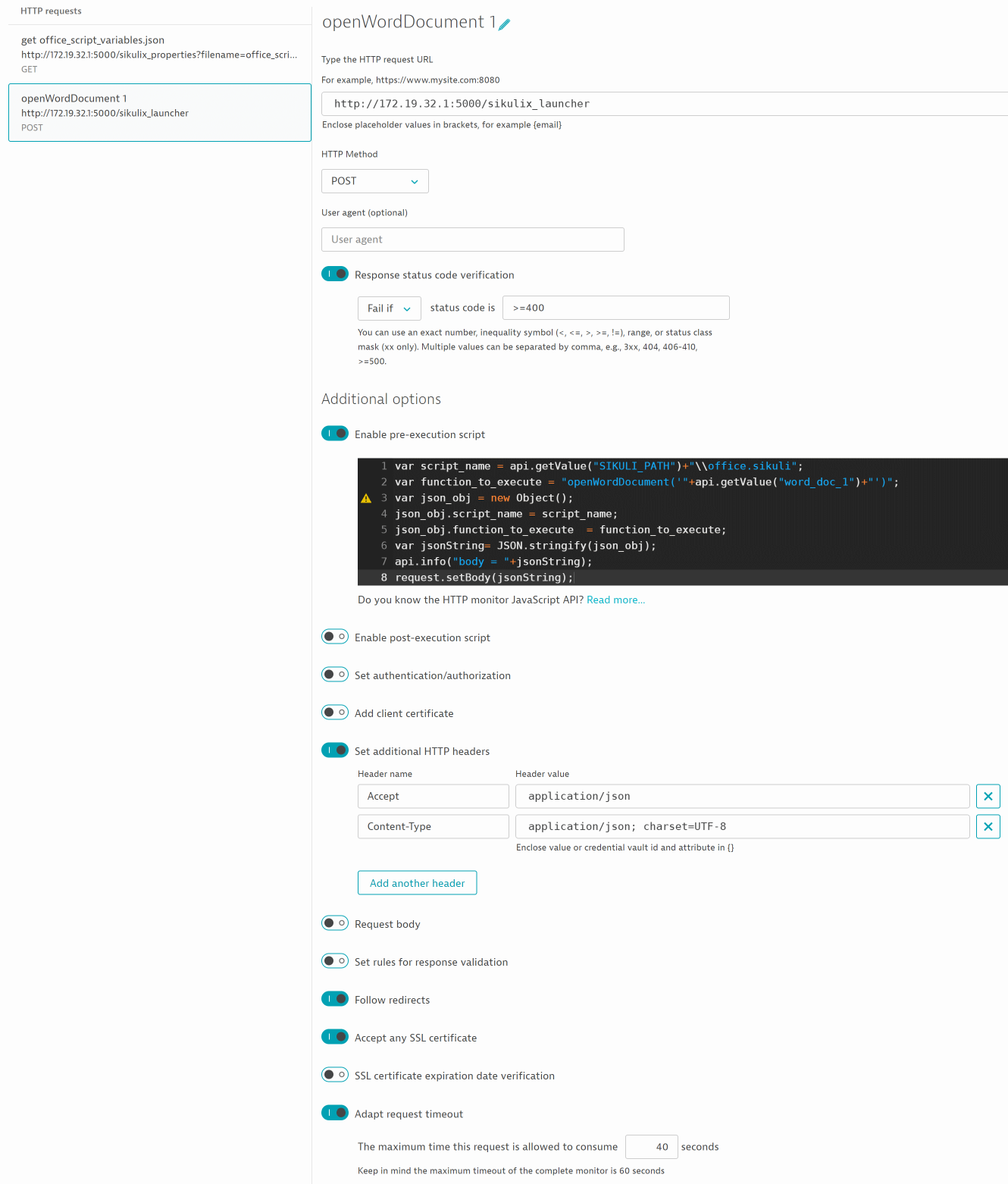


Note : the Sikulix IDE can be executed by double clicking on « sikulixide-2.0.5.jar », or by executing the following command :

* "c:\Program Files\dynatrace\synthetic\jre\bin\java.exe" -jar ./sikulixide-2.0.5.jar

### Dynatrace side

Below is an example HTTP Monitor step that will trigger the Sikulix script by calling the « /testtool\_launcher » POST URL :



A pre-execution srcript gets the placeholders from the previous step, builds the command line for Sikulix and sets the 2 POST parameters :

var script\_name = api.getValue("SIKULI\_PATH")+"\\office.sikuli";

var function\_to\_execute = "openWordDocument('"+api.getValue("word\_doc\_1")+"')";

var json\_obj = new Object();

json\_obj.script\_name = script\_name;

json\_obj.function\_to\_execute = function\_to\_execute;

var jsonString= JSON.stringify(json\_obj);

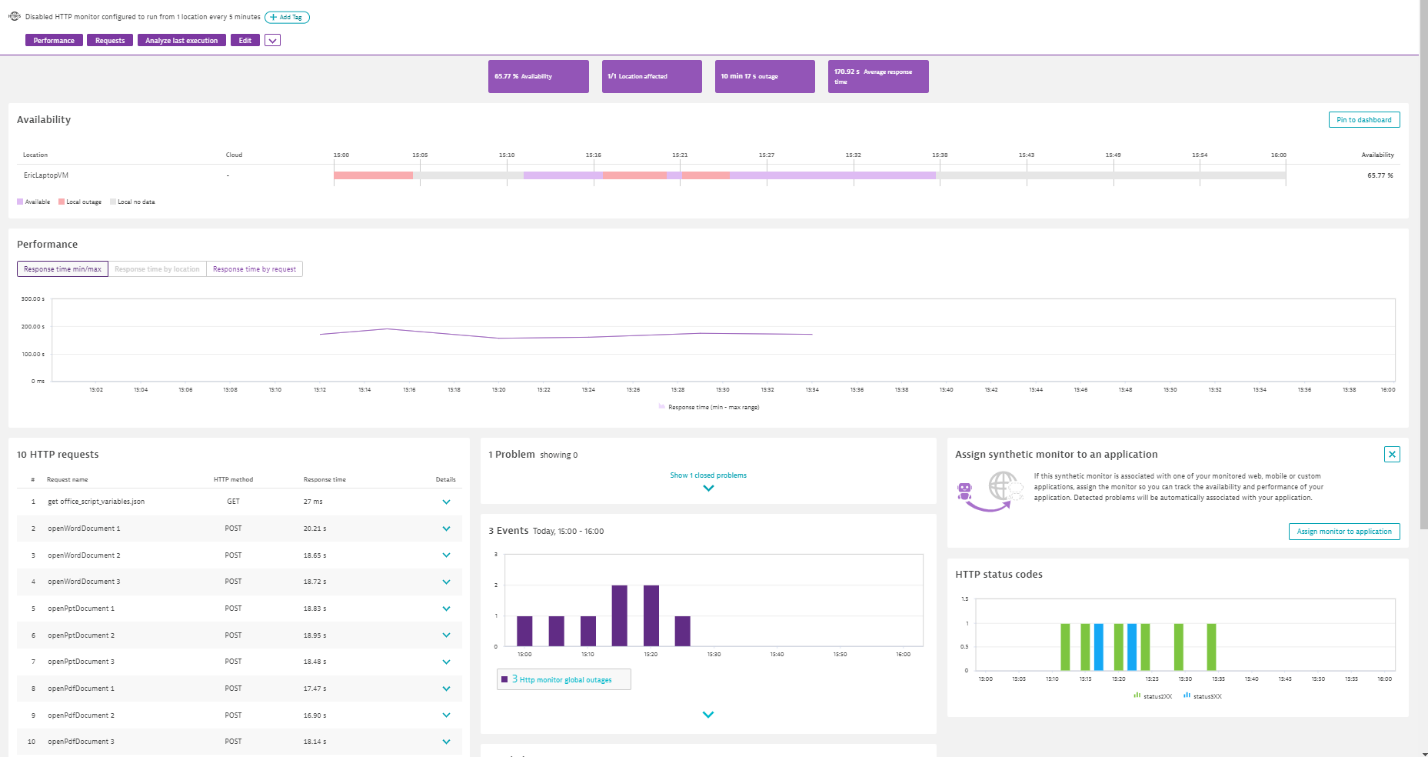
api.info("body = "+jsonString);

request.setBody(jsonString);

Note: as the POST content is sent as a json structure, make sure to set the appropriate request headers:

* Accept application/json
* Content-Type application/json; charset=UTF-8

Every time the Dynatrace http Monitor Synthetic script runs, say every 5 mn, it will report execution times and errors :



In case of error, the Sikulix error message is available in « Analyze last execution »  :

