Rudimentary Website Monitoring

with Selenium and PyVirtualDisplay

Michael Handler
LB Foster/Salient Systems, Inc.

Motivation

- We host servers for another business unit
- We monitor our servers and theirs with Zabbix
- They wanted a monitoring point for the health of their website
- Zabbix’s built in web monitoring facilities couldn’t do it

Library Choice

- https://elitedatascience.com/python-web-scraping-libraries
  - Requests
  - Beautiful Soup
  - Lxml
  - Selenium
  - Scrapy

Selenium

- Selenium is an interface to a browser driver
- Let’s you programmatically control a browser using a simple API
- Let’s you do what you need to do and check the results without needing to formulate http requests in detail
- One trade-off is performance – each session is firing up a browser
- Another trade-off is that finding the right version of the browser driver can be tricky

Installing Selenium and the browser driver

- pip install selenium
- http://selenium-python.readthedocs.io/installation.html#downloading-python-bindings-for-selenium
- https://github.com/mozilla/geckodriver/releases
- Getting the right version of the geckodriver can be an issue. I had to use two different versions on different machines because the version of Firefox on the server was different than the version on my workstation../on

DEMO

Monitoring script – Imports

```
from pyzabbix import ZabbixMetric, ZabbixSender
from pyvirtualdisplay import Display
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import logging
import sys
import time
import uuid
```

Monitoring script – main() part 1

```
def main():
    global logged_in

    # set up infrastructure -- virtual display and logging
    display = Display(visible=0, size=(800, 600))
    display.start()
    logging.basicConfig(filename="/tmp/RPM_website_check.log",
                                                    level=logging.INFO, format='%(asctime)s %(message)s')

    logging.info("Check RPM website...")



    logging.info("Connecting to browser...")
    driver = None
    try:
        driver = webdriver.Firefox()
    except Exception as e:
        cleanup_and_exit(driver, display, "Couldn't connect to browser: " + str(e),
NO_BROWSER_CONNECTION)

    logging.info("Connecting to site...")
    try:
        driver.get("http://www.lbfosterrpm.com")
    except Exception as e:
        cleanup_and_exit(driver, display, "Couldn't reach page: " + str(e), NOT_REACHED)
```

Monitoring script – main() part 2

```
logging.info("Preparing to log in...")
try:
    name_field = driver.find_element_by_name("tb_login")
    password_field = driver.find_element_by_name("tb_password")
    name_field.send_keys("<username>")
    password_field.send_keys("<password>")
    sign_in_button =  driver.find_element_by_name("b_login")
except Exception as e:
    cleanup_and_exit(driver, display, "Login setup failed: " + str(e),
                                                        LOGIN_SETUP_FAILED)

logging.info("Logging in...")
try:
    sign_in_button.click()
    time.sleep(10)
    if "Login" in driver.title:
        logging.info(driver.title)
        raise ValueError("Login failed")
except Exception as e:
    cleanup_and_exit(driver, display, str(e), LOGIN_FAILED)

logging.info("Login succeeded")
logged_in = True
```

Monitoring script – main() part 3

```
    try:
    logging.info("Waiting for anchors to load...")
    element = WebDriverWait(driver, 15).until(
        EC.presence_of_element_located((By.TAG_NAME, "a"))
    )
except Exception as e:
    cleanup_and_exit(driver, display, "Page contents not as expected: " + str(e), UNEXPECTED_PAGE_CONTENTS)


# Do some rudimentary checks to verify that the page has loaded normally
try:
    logging.info("Checking page contents...")
    table_title = driver.find_element_by_id("LblTitle")
    if table_title.text != "Units Dashboard":
        raise  ValueError("Table text is wrong, looking for 'Units Dashboard', found '" + table_title.text + "'")
    links = driver.find_elements_by_tag_name("a")
    milepost = False
    collector = False
    mode = False
    for elem in links:
        if elem.text == "Milepost":
            milepost = True
        if elem.text == "Collector":
            collector = True
        if elem.text == "Mode":
            mode = True
    if not (milepost and collector and mode):
        raise ValueError("Column headers not as expected")
except Exception as e:
    cleanup_and_exit(driver, display, "Page contents not as expected: " + str(e), UNEXPECTED_PAGE_CONTENTS)


# success
cleanup_and_exit(driver, display, "Success: Page contents verified", SUCCESS)
```

Monitoring script – all else

```
SUCCESS=0
NOT_REACHED = 1
LOGIN_SETUP_FAILED = 2
LOGIN_FAILED = 3
UNEXPECTED_PAGE_CONTENTS = 4
NO_BROWSER_CONNECTION = 5

logged_in = False

def logout(driver):
    logout_link = driver.find_element_by_id("LBLogout")
    logging.info("Logging out")
    logout_link.click()

def cleanup_and_exit(driver, display, message, zabbix_result):
    try:
        if driver != None:
            if zabbix_result != SUCCESS:
                driver.save_screenshot('/tmp/RPM_screenshot_' + str(uuid.uuid4()) + '.png')

            if logged_in:
                logout(driver)

            driver.close()
    except Exception:
        pass
    finally:
        logging.info(message)
        display.stop()

    packet = [ZabbixMetric('vtest', 'FM.web.checker', zabbix_result)]
    result = ZabbixSender(zabbix_server='zabbix-dc').send(packet)

    sys.exit(zabbix_result)
```

Screencap sending script (1)

```
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
from pathlib import Path
import logging
import os
import smtplib

from_addr = 'zabbix@salientsystems.com'
to_addrs = [<list of email recipients>]

LAST_SCREENCAP_FILE_PATH = '/tmp/FM_last_screencap_sent.tmp'

def main():

    logging.basicConfig(filename="/tmp/RPM_website_check.log", level=logging.INFO, format='%(asctime)s %(message)s')

    logging.info("Selecting screencap to send...")
    screencap_names = list(filter(lambda filename: filename.startswith("RPM_screenshot_"), os.listdir('/tmp')))
    screencap_infos = list(map(lambda screencap_name: ['/tmp/' + screencap_name,
os.stat('/tmp/'+screencap_name)],screencap_names))

    newest_sc_time = 0
    newest_sc_name = None
    for info in screencap_infos:
        if info[1].st_mtime > newest_sc_time:
            newest_sc_time = info[1].st_mtime
            newest_sc_name = info[0]

    last_sent_info = None
    lsf = Path(LAST_SCREENCAP_FILE_PATH)
    if (lsf.exists()):
        last_sent_info = os.stat(LAST_SCREENCAP_FILE_PATH)
```

Screencap sending script (2)

```
if (newest_sc_name != None and (last_sent_info == None or last_sent_info.st_mtime < newest_sc_time)):
    logging.info("Sending screencap: " + newest_sc_name)
    msg = MIMEMultipart()
    msg['Subject'] = 'Screencap from most recent RPM Web Check alarm'
    msg['From'] = from_addr
    msg['To'] = ", ".join(to_addrs)

    fp = open(newest_sc_name, 'rb')
    img = MIMEImage(fp.read())
    fp.close()
    msg.attach(img)

    s = smtplib.SMTP('smtpdc.salientsystems.com')
    s.sendmail(from_addr, to_addrs, msg.as_string())
    s.quit()

    # keep a record of the last time we sent a screencap
    with open(LAST_SCREENCAP_FILE_PATH, "w") as f:
        f.write("")
else:
    logging.info("No screencap to send")
```