Python Test Setup Example

This script provides an example of how you might configure your own automated tests to run in the Sauce Labs browser cloud. The sample test uses environment variables for authentication, assigns a tag and build number for test result management, and reports Pass/Fail status to the Sauce Labs dashboard.

> ⚠ **Example Only**
>
> The code in this topic is presented as an example only, since your tests and testing environments may require specialized scripting. This information should be taken only as an illustration of how you could set up your tests with Sauce Labs, and is not directly supported by Sauce.

Prerequisites

- You should have a Sauce Labs account
- You should have a Selenium environment with the bindings for Python set up on the local machine where you'll launch your tests. If you need help with this, check out the ReadMe in the Python example scripts repository on GitHub.
- You should read the Best Practices for Running Tests
- You will need to install the Selenium WebDriver client driver to your local Python environment
  You can either download the driver from the link, or use `pip` to install it.
- You can also optionally install the Sauce Python client, which provides features for reporting job information to the Sauce Labs dashboard.

Setup Example Script

The sample test opens the browser, navigates to the saucelabs.demo web app, and then closes the browser.

To run this script against your own app using your Sauce Labs credentials:

1. Set your authentication credentials as environment variables to connect the test to your Sauce Labs account.
2. Enter the URL for the web app you want to test in the place of `saucedemo.com`.

Once you've been able to run the test against your web app, check out the Platform Configurator to see more of the desired capabilities you can use when testing with Sauce.

You can clone this script from the `saucelabs-training` repository on GitHub: https://github.com/saucelabs-training/demo-python

There are examples using both `pytest` and `unittest` frameworks, change directories to the relevant module before running your tests:

- https://github.com/saucelabs-training/demo-python/blob/master/on-boarding-modules/pytest-examples/test_module4_pytest.py

test_module4_pytest.py (source from Sauce Labs Training Python)

test_module4_pytest.py

```
import pytest
import os
from selenium import webdriver
from _pytest.runner import runtestprotocol

@pytest.fixture
def driver(request):
    sauce_username = os.environ["SAUCE_USERNAME"]
    sauce_access_key = os.environ["SAUCE_ACCESS_KEY"]
    remote_url = "https://ondemand.saucelabs.com:443/wd/hub"

    sauceOptions = {
        'screenResolution': '1280x768',
        'seleniumVersion': '3.141.59',
        # best practices involve setting a build number for version control
        'build': 'Onboarding Sample App - Python + Pytest',
        'name': '4-best-practices',
        'username': sauce_username,
        'accessKey': sauce_access_key,
        # tags to filter test reporting.
        'tags': ['instant-sauce', 'pytest', 'module4'],
        # setting sauce-runner specific parameters such as timeouts helps
        # manage test execution speed.
        'maxDuration': 1800,
        'commandTimeout': 300,
        'idleTimeout': 1000,
    }

    chromeOpts = {
        'platformName': 'Windows 10',
        'browserName': 'chrome',
        'browserVersion': 'latest',
        'goog:chromeOptions': {'w3c': True},
        'sauce:options': sauceOptions
    }

    browser = webdriver.Remote(command_executor=remote_url, desired_capabilities=chromeOpts)
    yield browser
    browser.quit()


def test_should_open_chrome(driver):
    driver.get("https://www.saucedemo.com")
    actual_title = driver.title
    expected_title = "Swag Labs"
    assert expected_title == actual_title
```

- https://github.com/saucelabs-training/demo-python/blob/master/on-boarding-modules/unittest-examples/test_module4_unittest.py

test_module4_unittest.py (source from Sauce Labs Training Python)

test_module4_unittest.py

```
# Selenium 3.14+ doesn't enable certificate checking
import unittest
import os
from selenium import webdriver

sauce_username = os.environ["SAUCE_USERNAME"]
sauce_access_key = os.environ["SAUCE_ACCESS_KEY"]
remote_url = "https://ondemand.saucelabs.com:443/wd/hub"

class Module4Test(unittest.TestCase):

    def setUp(self):
        sauceOptions = {
            'screenResolution': '1280x768',
            'seleniumVersion': '3.141.59',
            'build': 'Onboarding Sample App - Python + UnitTest',
            'name': '4-best-practices',
            'username': sauce_username,
            'accessKey': sauce_access_key,
            # tags to filter test reporting.
            'tags': ['instant-sauce', 'ruby-rspec', 'module4'],
            # setting sauce-runner specific parameters such as timeouts helps
            # manage test execution speed.
            'maxDuration': 1800,
            'commandTimeout': 300,
            'idleTimeout': 1000
        }
        # In ChromeOpts, we define browser and/or WebDriver capabilities such as
        # the browser name, browser version, platform name, platform version
        chromeOpts = {
            'platformName': 'Windows 10',
            'browserName': 'chrome',
            'browserVersion': 'latest',
            'goog:chromeOptions': {'w3c': True},
            'sauce:options': sauceOptions
        }
        self.driver = webdriver.Remote(command_executor=remote_url,
desired_capabilities=chromeOpts)

    def test_should_open_chrome(self):
        # Substitute 'http://www.saucedemo.com' for your own application URL
        self.driver.get("https://www.saucedemo.com")
        assert ("Swag Labs" in self.driver.title)

    def tearDown(self):
        if self.driver.title == 'Swag Labs':
            self.driver.execute_script('sauce:job-result=passed')
        else:
            self.driver.execute_script('sauce:job-result=failed')
        self.driver.quit()


if __name__ == '__main__':
    unittest.main()
```

Testing with a Proxy

If you're trying to run this script from behind a VPN or a corporate proxy, you must use either IPSec or Sauce Connect Proxy. Once you've downloaded and installed the relevant software, add the following capability to the test script:

```
'tunnelIdentifier': '<tunnel_id>',
```

Running the Test

1. Navigate to the root project directory and use `pip` to install the latest Selenium library for use in the script:

```
$ pip install -r requirements.txt
```

2. Set your Sauce Labs Credentials as envrionment variables, indicated by the following lines in the script:

test_module4_pytest.py (source from Sauce Labs Training Python)

```
    sauce_username = os.environ["SAUCE_USERNAME"]
    sauce_access_key = os.environ["SAUCE_ACCESS_KEY"]
```

3. Depending on which framework you're using, your commands may be different to run the tests. Use any of the following command based on the chosen framework:

pytest:

```
pytest on-boarding-modules/pytest-examples/test_module4_pytest.py
```

unittest:

```
 python -m unittest on-boarding-modules/unittest-examples/test_module4_unittest.py
```