МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Технологии автоматизации процесса разработки программного обеспечения»

Тема: Использование Docker

Вариант 18

Студент гр. 8346 _____ Фёдоров И.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Фёдоров И.В.

Группа 8346

Тема работы: Использование Docker

Исходные данные:
Требуется реализовать конфигурацию docker-compose, состоящую из двух контейнеров – с приложением и с тестами.

Содержание пояснительной записки:

- Содержание
- Введение
- Постановка задачи
- Описание Dockerfile
- Описание скриптов запуска тестов
- Описание конфигурации docker-compose
- Заключение
- Список использованных источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Фёдоров И.В.

Преподаватель _____ Заславский М.М.

АННОТАЦИЯ

В данной работы была представлена реализация конфигурации docker-compose, описывающая два сервиса, а также докерфайлы, описывающие сборку образов и в последствии контейнеров указанных сервисов. В первом контейнере запускается приложение, во втором тестовая среда. Тестирование включает анализа кода pylint и html linter, запуска статического, интеграционного и selenium тестов.

SUMMARY

In this work, an implementation of the docker-compose configuration was presented, describing two services, as well as docker files describing the assembly of images and, subsequently, containers of these services. In the first container, the application is launched, in the second, the test environment. Testing includes analyzing pylint and html linter code, running static, integration and selenium tests.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В этой работе была реализована docker-compose конфигурация, которая включает в себя два сервиса и соответствующие dockerfile для их сборки. Один контейнер предназначен для запуска веб-сервера, в то время как другой предназначен для запуска тестов. Тестирование включает анализ кода при помощи pylint и html linter, а также интеграционного и selenium тестов.

1. ПОСТАНОВКА ЗАДАЧИ

Необходимо реализовать docker-compose конфигурацию из двух контейнеров:

- app – контейнер с существующим демо-проектом;
- tester – контейнер с запуском всех тестов.

Для достижения данной задачи необходимо написать два файла Dockerfile соответственно, которые удовлетворяют следующим требованиям:

- Минимальная версия докера Docker version 19.03.13, build 4484c46d9d;
- Базовый образ ubuntu:22.04;
- Не использовать Expose;
- При установке любых пакетов и программ (в том числе в requirements) ВСЕГДА указывать версии;
- Ограничить установку зависимостей apt одной строкой (один RUN);
- Если настройка одной части приложения состоит из нескольких команд → необходимо разместить их в одном слое (в одном RUN).

Данные контейнеры должны подниматься посредством docker-compose, который должен:

- Минимальная версия docker compose version 1.27.4, build 40524192;
- Все должно собирается по команде docker-compose build без sudo;
- Не использовать тип сети HOST;
- Не отрывать лишних (непредусмотренных заданием) портов;
- Не использовать порты хост-машины ⇐1024.

В соответствии с вариантом задания необходимо выполнить следующие задачи:

- Проверка бьютификатором для HTML;
- Анализ по 10 существующим критериям;
- Создание интеграционного тест, проверяющего загрузку файла;

- Создание теста используя Selenium, который должен проверять правку html кода страницы;
- Добавить внешний доступ по SSH для контейнеров app и tester по паролю;
- Каждый этап тестирования - в docker log (stdout + stderr) + добавить к записям лога timestamp;
- Возможность передачи списка этапов тестирования для запуска с помощью файла .env;
- Ограничить Максимальное Количество процессов.

2. ОПИСАНИЕ DOCKERFILE

2.1. Контейнер app

В качестве базового образа используется ubuntu 22.04.

В начале создания контейнера происходит установка apt-зависимостей. Git необходим для получения кода приложения, python – для запуска приложения, pip – для установки необходимых для запуска библиотек, pwgen – для генерации пароля для SSH-входа, openssh-server – для обеспечения доступа по SSH.

Затем происодит настройка контейнера для поддержки возможности запуска ssh-сервера.

Затем копируется файл со списком python-библиотек и происходит их установка с помощью pip.

После чего происходит получение кода приложения, копирования скрипта, в котором создаетс пароль и запускается ssh-сервер, и применение патча, обеспечивающего доступ к приложению снаружи контейнера и поддержку флага отладки.

В качестве основного процесса контейнера используется предоставленное приложение.

2.2. Контейнер tester

В качестве базового образа используется ubuntu 22.04.

В начале создания контейнера происходит установка apt-зависимостей. Git необходим для получения кода приложения, python – для запуска тестов, pip – для установки необходимых для запуска библиотек, wget – для скачивания firefox и geckodriver, xvfb – для запуска firefox без графического интерфейса, libgtk, libdbus и libasound – зависимости, необходимые для запуска firefox.

Затем происходит скачивание и установка firefox и geckodriver, они необходимы для запуска selenium-тестов.

После чего происходит получение кода приложения, который необходим для проверки с помощью pylint и html linter.

Затем в контейнер добавляется файл с зависимостями, необходимыми для запуска тестов и их установка.

Далее происходит копирование кода самих тестов и скрипта, для запуска всех тестов.

В качестве основного процесса используется http-сервер, входящий в базовую поставку python3.

3. ОПИСАНИЕ СКРИПТОВ И ТЕСТОВ

3.1. Описание test_all.sh

Данный скрипт запускает все тест.

Переменная root присваивает путь к директории скрипта.

Затем выводится строка с заголовком Run pipeline в красном цвете.

Далее определены четыре функции: start_pylint, start_html_linter, start_selenium_tests, star_tintegration_tests, каждая из которых запускает соответствующий тестовый скрипт в поддиректории. В конце каждой функции присваивается результат выполнения команды.

Функция get_report определяет результаты выполнения всех тестов и выводит их на экран в красном или зеленом цвете в зависимости от того, пройден ли тест.

В конце скрипта используется условный оператор case для определения, какой тест нужно запустить. Если передан ключ -p, будет выполнен только тест PyLint, если передан ключ -h - только HTML linter test, если передан ключ -s - только Selenium test, если передан ключ -i - только Integration tests. Если ни один ключ не передан, будут выполнены все тесты и выведен отчет.

3.2. Описание style_test.sh

Данный скрипт выполняет тестирование файлов с расширением .py на соответствие стандартам кодирования с помощью инструмента PyLint.

Сначала выполняется команда sleep 1 - она задерживает выполнение скрипта на 1 секунду.

Затем с помощью команды find ищутся файлы с расширением .py в текущей директории и во всех поддиректориях. Результат поиска передается команде xargs, которая запускает PyLint с файлом конфигурации ./tests/.pylintrc на найденных файлах.

После выполнения команды в переменную status записывается код выполнения команды PyLint. Если код равен 0, то выводится сообщение "Pylint:

PASSED" и скрипт завершает работу с кодом 0. В противном случае выводится сообщение "Pylint: FAILED" и скрипт завершает работу с кодом 1.

Таким образом, данный скрипт проверяет файлы с расширением .py на соответствие стандартам кодирования с помощью PyLint и выводит соответствующий статус.

3.3. Описание static_test.sh

Скрипт начинается со встроенной команды "sleep 1", что означает задержку выполнения скрипта на 1 секунду.

Затем исполняется команда "annotate-output "+%D %H:%M:%S" html_lint.py ./templates/*.html". Эта команда запускает скрипт html_lint.py с аргументами и дополнительно аннотирует вывод. Аннотация вывода позволяет определить, какая строка вывода была сгенерирована с какой команды.

Исполнение команды завершится с выходным кодом, который будет сохранен в переменную $status.

Далее в скрипте проверяется значение переменной статуса. Если $status равен 0, то выводится сообщение "Html lint: PASSED" и скрипт завершается с кодом возврата 0 (успешное выполнение). Если $status не равен 0, то выводится сообщение "Html lint: FAILED" и скрипт завершается с кодом возврата 1 (неуспешное выполнение).

3.4. Описание selenium_test.sh

Данный sh скрипт начинается с команды "echo "Start xvfb"", которая просто выводит сообщение в терминал.

Затем запускается X-сервер командой "Xvfb :1 -screen 0 1024x768x16 &> xvfb.log &", который запускается на первом дисплее с соответствующим разрешением экрана. Весь вывод этой команды сохраняется в файле xvfb.log. Команда запускается в фоновом режиме (символ & в конце).

Далее устанавливается значение переменной $DISPLAY на :1.0, что означает, что эта сессия X будет использоваться для запуска тестов.

Далее скрипт задерживается на 1 секунду командой "sleep 1" для полной загрузки и инициализации X-сервера.

Далее запускается тестирование с помощью pytest и указанными параметрами. Отчеты о тестировании логируются в формате, указанном после параметра --log-format. Значение выходного кода сохраняется в переменную $status.

Далее происходит проверка переменной статуса. Если $status равен 0, то выводится сообщение "Selenium: PASSED", происходит убийство X-сервера командой "kill $(pgrep -f Xvfb)" и выводится сообщение "Stop xvfb".

Если $status не равен 0, то выводится сообщение "Selenium: FAILED", происходит убийство X-сервера командой "kill $(pgrep -f Xvfb)" и выводится сообщение "Stop xvfb". Скрипт завершается с кодом возврата 1 (неуспешное выполнение).

3.5. Описание integration_test.sh

Данный sh скрипт начинается со встроенной команды "sleep 1", что означает задержку выполнения скрипта на 1 секунду.

Затем запускается pytest для проведения интеграционного тестирования, указанными параметрами. Отчеты о тестировании логируются в формате, указанном после параметра --log-format. Значение выходного кода сохраняется в переменную $status.

Далее происходит проверка переменной статуса. Если $status равен 0, то выводится сообщение "Integration: PASSED" и скрипт завершается с кодом возврата 0 (успешное выполнение). Если $status не равен 0, то выводится сообщение "Integration: FAILED" и скрипт завершается с кодом возврата 1 (неуспешное выполнение).

3.6. Описание start_ssd.sh

Данный sh скрипт начинается с генерации случайного пароля с помощью команды "pwgen -n 16 -1 -s". Результат сохраняется в переменную $PASSWORD.

Затем с помощью команды "echo root:$PASSWORD | chpasswd" происходит изменение пароля пользователя root на сгенерированный выше.

После этого выводится сообщение "SSH Password: $PASSWORD", где $PASSWORD - сгенерированный пароль.

И наконец, запускается ssh-сервер командой "/usr/sbin/sshd".

3.7. Описание integration_test.py

Класс TestIntegration, проверяет возможность загрузки и скачивания файла на сервере по заданному URL.

В начале кода задаются константы URL, FILEPATH и FILENAME, которые используются в дальнейшем.

Затем определен метод test_upload_download, который содержит следующие шаги:

1. Открытие файла с именем FILENAME из директории FILEPATH в бинарном режиме и чтение его содержимого в переменную content_original.

2. Открытие того же файла в бинарном режиме и отправка его содержимого на сервер по URL с помощью метода POST из библиотеки requests.

3. Получение содержимого файла с именем FILENAME с сервера по URL с помощью метода GET из библиотеки requests и сохранение его в переменную download.

4. Чтение первых 1000 байт из полученного содержимого файла и сохранение их в переменную content_downloaded.

5. Сравнение содержимого оригинального файла content_original и загруженного файла content_downloaded.

Если содержимое файлов одинаково, то тест пройден успешно, иначе возникнет ошибка AssertionError.

3.8. Описание selenium_test.py

Класс TestSelenium, проверяет возможность добавления на страницу кнопки с заданным текстом.

В начале кода задается константа URL, которая используется в дальнейшем.

Затем определен метод setup_method, который создает экземпляр браузера Firefox с помощью библиотеки selenium и заданными параметрами.

Далее определен метод test_custom_button, который содержит следующие шаги:

1. Открытие страницы по URL с помощью метода get из библиотеки selenium.

2. Добавление на страницу кнопки с заданным текстом с помощью метода execute_script из библиотеки selenium.

3. Ожидание одной секунды для обновления страницы.

4. Поиск добавленной кнопки с помощью метода find_element из библиотеки selenium и проверка ее текста на соответствие заданному значению.

Если текст кнопки соответствует заданному значению, то тест пройден успешно, иначе возникнет ошибка AssertionError.

В конце определен метод teardown_method, который закрывает браузер после выполнения тестов.

4. ОПИСАНИЕ КОНФИГУРАЦИИ DOCKER-COMPOSE

Данный файл описывает два сервиса приложения: app и tester.

Параметр version определяет используемую версию формата файла.

Для сервиса app определяются порты, которые будут проксироваться - "127.0.0.1:5000:5000" и "127.0.0.1:5022:22".

Затем указывается, что для этого сервиса необходимо использовать Dockerfile_app из директории ./app.

Далее в разделе deploy определяются ограничения для ресурсов, запрашиваемых приложением: максимальное количество процессов, которые могут быть запущены - 18.

В конце указывается, что для сервиса app нужно использовать алиас 8346_fedorov_iv-app-1 и подключить этот сервис к стандартной сети Docker.

Для сервиса tester определяется порт "127.0.0.1:5023:22".

Затем указывается, что для этого сервиса необходимо использовать Dockerfile_tester из директории ./test.

В конце указывается, что для сервиса tester нужно использовать алиас 8346_fedorov_iv-tester-1 и подключить этот сервис к стандартной сети Docker.

Оба сервиса используют одну и ту же сеть Docker (default), но имеют различные алиасы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были изучены возможности docker и docker-compose. Была реализована конфигурация, включающая в себя два сервиса. Также при реализации были соблюдены поставленные ограничения и выполнены требования.

В ходе работы были реализованы различные виды тестов, доступ к контейнерам через SSH, ограничение максимального количества процессов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по Dockerfile // Docker docs. URL: https://docs.docker.com/engine/reference/builder/ (дата обращения: 30.4.2023).
2. Документация по Docker-compose // Docker docs. URL: https://docs.docker.com/compose/compose-file/ (дата обращения: 30.4.2023).
3. Документация pytest // Full pytest documentation. URL: https://docs.pytest.org/en/7.1.x/contents.html (дата обращения: 23.4.2023).
4. Документация pylint // Pylint User Manual. URL: https://docs.pylint.org (дата обращения: 30.4.2023).
5. Документация Selenium // The Selenium Browser Automation Project. URL: https://www.selenium.dev/documentation/ (дата обращения: 30.4.2023).

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Файл docker-compose.yml:

```
version: "3"
services:
  app:
    ports:
      - "127.0.0.1:5000:5000"
      - "127.0.0.1:5022:22"
    build:
      context: ./app
      dockerfile: Dockerfile_app
    deploy:
      resources:
        limits:
          pids: 18
    networks:
      default:
        aliases:
          - 8346_fedorov_iv-app-1
  tester:
    ports:
      - "127.0.0.1:5023:22"
    build:
      context: ./test
      dockerfile: Dockerfile_tester
    networks:
      default:
        aliases:
          - 8346_fedorov_iv-tester-1
```

Файл Dockerfile_app:

```
FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \
    openssh-server=1:8.9p1-3ubuntu0.1 \
    pwgen=2.08-2build1 \
    python3-pip=22.0.2+dfsg-1ubuntu0.2 \
    python3=3.10.6-1~22.04 \
    git=1:2.34.1-1ubuntu1.9 \
    patch=2.7.6-7build2 \
    && rm -rf /var/lib/apt/lists/*

RUN mkdir -p /var/run/sshd \
    && sed -i '/PermitRootLogin prohibit-password/c\PermitRootLogin yes' /etc/ssh/sshd_config

RUN git clone --depth=1 https://github.com/moevm/devops-examples.git
WORKDIR /devops-examples/EXAMPLE_APP/
```

```
COPY requirements.txt add_host.patch start_sshd.sh ./
RUN pip3 install -r requirements.txt \
   && patch main.py add_host.patch

ENTRYPOINT ["python3", "main.py"]
```

Файл Dockerfile_tester:

```
FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \
   openssh-server=1:8.9p1-3ubuntu0.1 \
   pwgen=2.08-2build1 \
   python3=3.10.6-1~22.04 \
   python3-pip=22.0.2+dfsg-1ubuntu0.2 \
   git=1:2.34.1-1ubuntu1.9 \
   wget=1.21.2-2ubuntu1 \
   xvfb=2:21.1.4-2ubuntu1.7~22.04.1 \
   devscripts=2.22.1ubuntu1 \
   libgtk-3-0=3.24.33-1ubuntu2 \
   libdbus-glib-1-2=0.112-2build1 \
   libasound2=1.2.6.1-1ubuntu1\
   && rm -rf /var/lib/apt/lists/*

RUN mkdir -p /var/run/sshd \
   && sed -i '/PermitRootLogin prohibit-password/c\PermitRootLogin yes' /etc/ssh/sshd_config

RUN wget -O firefox-setup.tar.bz2 "https://download.mozilla.org/?product=firefox-
latest&os=linux64" \
   && tar -xvjf firefox-setup.tar.bz2 \
   && ln -s /usr/local/firefox/firefox /usr/bin/firefox \
   && wget https://github.com/mozilla/geckodriver/releases/download/v0.32.2/geckodriver-
v0.32.2-linux32.tar.gz \
   && tar -xvzf geckodriver* \
   && cp geckodriver /usr/bin/geckodriver

RUN git clone --depth=1 https://github.com/moevm/devops-examples.git
WORKDIR devops-examples/EXAMPLE_APP/

ADD requirements_test.txt start_sshd.sh ./
RUN pip3 install -r requirements_test.txt

COPY tests tests

ENTRYPOINT ["python3", "-m", "http.server", "3000"]
```

Файл add_host.patch:

```
78c78
<     app.run(debug = True)
---
>     app.run(host='0.0.0.0', debug = True)
```

Файл .pylintrc:

```
[MAIN]
ignore=tests

[MESSAGES CONTROL]
disable=all
enable=main, basic, format, classes, design, imports, typecheck, variables, refactoring, string

[REPORTS]
output-format=colorized
reports=no
evaluation= 10.0 -((float(5 * error + warning + refactor + convention) / statement) * 10)
msg-template={C}: {line}: {msg} ({symbol})
score=yes
```

Файл start_sshd.sh:

```
PASSWORD=$(pwgen -n 16 -1 -s)
echo root:$PASSWORD | chpasswd
echo "SSH Password: $PASSWORD"
/usr/sbin/sshd
```

Файл html-linter.json

```
{
 "files": [
  "**/*.html"
 ],
 "indentation": {
  "char": "space",
  "number": 2
 },
 "attributes": {
  "quotes": "double",
  "whitespace": 0,
  "vertical-align": true
 }
}
```

Файл test_all.sh

```
#!/bin/sh

root=$(dirname $0)


echo "\033[31m===============\nRun pipeline\n===============\033[0m"

start_pylint() {
   echo "\n\033[35m===============Run PyLint for 10 categories
test===============\033[0m\n"
```

```
    bash ${root}/style_test.sh
    style=$(echo $?)
}

start_html_linter() {
    echo "\n\033[35m================ Run HTML linter test ================\033[0m\n"
    bash ${root}/static_test.sh
    static=$(echo $?)
}
start_selenium_tests() {
    echo "\n\033[35m================ Run Selenium tests ================\033[0m\n"
    bash ${root}/selenium_test.sh
    selenium=$(echo $?)
}

start_integration_tests() {
    echo "\n\033[35m================ Run Integration tests ================\033[0m\n"
    bash ${root}/integration_test.sh
    integration=$(echo $?)
}

get_report() {
  echo "\033[031m================\n Results \n================\033[0m"

  if [ "$style" -eq 0 ]
  then
     echo "\033[32mStyle: PASSED\033[0m"
  else
     echo "\033[31mStyle: FAILED\033[0m"
  fi

  if [ "$static" -eq 0 ]
  then
     echo "\033[32mStatic: PASSED\033[0m"
  else
     echo "\033[31mStatic: FAILED\033[0m"
  fi

  if [ "$selenium" -eq 0 ]
  then
     echo "\033[32mSelenium: PASSED\033[0m"
  else
     echo "\033[31mSelenium: FAILED\033[0m"
  fi

  if [ "$integration" -eq 0 ]
  then
     echo "\033[32mIntegration: PASSED\033[0m"
  else
     echo "\033[31mIntegration: FAILED\033[0m"
  fi
}
```

```
case "$1" in
  -p)
      start_pylint
      ;;
  -h)
      start_html_linter
      ;;
  -s)
      start_selenium_tests
      ;;
  -i)
      start_integration_tests
      ;;
  *)
      echo "\n\033[31m================ Run all tests ================\033[0m\n"
      start_pylint
      start_html_linter
      start_selenium_tests
      start_integration_tests
      echo "\n\033[31m================ End all tests ================\033[0m\n"
      get_report
      ;;
esac
```

Файл style_test.sh

```
#!/bin/bash

sleep 1

find . -type f -name "*.py" -print0 | xargs -0 pylint --rcfile=./tests/.pylintrc
status=$?

if [ $status -eq 0 ]
then
  echo "Pylint: PASSED"
  exit 0
else
  echo "Pylint: FAILED"
  exit 1
fi
```

Файл static_test.sh

```
#!/bin/bash

sleep 1

annotate-output "+%D %H:%M:%S"  html_lint.py ./templates/*.html
status=$?

if [ $status -eq 0 ]
```

```
then
   echo "Html lint: PASSED"
  exit 0
else
  echo "Html lint: FAILED"
  exit 1
fi
```

Файл selenium_test.sh

```
#!/bin/bash

echo "Start xvfb"
Xvfb :1 -screen 0 1024x768x16 &> xvfb.log  &

DISPLAY=:1.0
export DISPLAY

sleep 1

pytest --log-format="%(asctime)s %(levelname)s %(message)s" -v -rfpP --tb=no
./tests/selenium_test.py --disable-warnings --color=yes
status=$?

if [ $status -eq 0 ]
then
  echo "Selenium: PASSED"
  kill $(pgrep -f Xvfb)
  echo "Stop xvfb"
  exit 0
else
  echo "Selenium: FAILED"
  kill $(pgrep -f Xvfb)
  echo "Stop xvfb"
  exit 1
fi
```

Файл integration_test.sh

```
#!/bin/bash
sleep 1

pytest --log-format="%(asctime)s %(levelname)s %(message)s" -v -rfpP --tb=no
./tests/integration_test.py --disable-warnings --color=yes
status=$?

if [ $status -eq 0 ]
then
   echo "Integration: PASSED"
  exit 0
else
  echo "Integration: FAILED"
  exit 1
fi
```

Файл integration_test.py:

```
import requests

URL = "8346_fedorov_iv-app-1:5000"
FILEPATH = 'tests/'
FILENAME = 'test_file.txt'

class TestIntegration:
    def test_upload_download(self):
        with open(FILEPATH + FILENAME, 'rb') as file:
            content_original = file.read()

        with open(FILEPATH + FILENAME, 'rb') as file:
            requests.post(f'http://{URL}/upload', files={'file': file})

        download = requests.get(f'http://{URL}/download/{FILENAME}')
        content_downloaded = next(download.iter_content(1000))

        assert content_original == content_downloaded
```

Файл selenium_test.py:

```
from seleniumwire import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.service import Service
import time

URL = "8346_fedorov_iv-app-1:5000"

class TestSelenium:
    def setup_method(self):
        option = webdriver.FirefoxOptions()
        option.binary_location = '/firefox/firefox'
        driverService = Service('/geckodriver')
        self.driver = webdriver.Firefox(service=driverService, options=option)

    def test_custom_button(self):
        self.driver.get(f"http://{URL}/")
        time.sleep(1)
        self.driver.execute_script(
            "document.getElementById('timeout_buttons').innerHTML = '<button
id=\"custom_btn\">custom button</button>'")
        time.sleep(1)
        assert self.driver.find_element(By.ID, "custom_btn").get_attribute('innerHTML') == 'custom
button'

    def teardown_method(self):
        self.driver.close()
```