

Informe Segundo Parcial

Universidad ICESI

Curso: Sistemas Operativos

Estudiante: Paula Andrea Bolaños Arias.

Código: 13207002 – A00068008

Correo: pauandre27@gmail.com

Objetivos

• Diseñar e implementar pruebas para servicios web que realizan acciones sobre el sistema operativo. • Emplear un servidor de integración continua para la realización de pruebas automáticas sobre servicios web.

Introducción

En el presente informe se explicará una guía paso a paso de cómo realizar pruebas unitarias para servicios web, usando jenkins.

Prerrequisitos

• Sistema operativo CentOS 6.8 versión servidor • Servidor de Integración continua Jenkins

Instalación

Teniendo en cuenta que, para la implementación de los servicios web es necesario instalar jenkins, aquí está un tutorial que puede ser útil:

<https://www.youtube.com/watch?v=Jy6NfzIVAKg> (<https://www.youtube.com/watch?v=Jy6NfzIVAKg>)

Desarrollo

1. Cree un usuario llamado developer

```
# adduser developer
# passwd developer
```

2. Dé al usuario developer los permisos de root

nano /etc/sudoers

Añada a la línea debajo de “root ALL=(ALL:ALL)ALL”, la línea: “developer ALL=

(ALL:ALL)ALL”.

3. Conozca su ip para las diferentes conexiones remotas y para las pruebas, recuerde haber subido las interfaces

ifconfig

4. Cree un directorio donde almacenar su proyecto

```
> # su developer
$ mkdir ~/projects/myproject
$ cd ~/projects/myproject
```

5. Cree sus archivos .py con el código y las pruebas

```
> $ nano files.py
Escriba aquí lo siguiente:
from flask import Flask, abort, request
import json

from files_commands import get_all_files, add_file, remove_file

app = Flask(__name__)
api_url = '/v1.0'

@app.route(api_url+' /files', methods=['POST'])
def create_file():
    content = request.get_json(silent=True)
    filename = content['filename']
    content = content['content']
    if not filename or not content:
        return "empty filename or content", 400
    if filename in get_all_files():
        return "file already exist", 400
    if add_file(filename, content):
        return "CREATED", 201
    else:
        return "error while creating file", 400

@app.route(api_url+' /files', methods=['GET'])
def read_files():
    list = {}
    list["files"] = get_all_files()
    return json.dumps(list), 200

@app.route(api_url+' /files', methods=['PUT'])
def update_file():
    return "NOT FOUND", 404
```

```

@app.route(api_url+' /files',methods=['DELETE'])
def delete_files():
    error = False
    for filename in get_all_files():
        if not remove_file(filename):
            error = True

    if error:
        return 'some files were not deleted', 400
    else:
        return 'OK. FILES DELETED', 200

if __name__ == "__main__":
    app.run(host='192.168.56.101',port=9090,debug='True')

$ nano files_commands.py

from subprocess import Popen, PIPE

def get_all_files():
    grep_process = Popen(["ls"], stdout=PIPE, stderr=PIPE)
    file_list = Popen(["awk", '{print $1}'], stdin=grep_process.stdout, stdout=PIPE, stderr=PIPE).communicate()[0].split('\n')
    return filter(None,file_list)

def add_file(filename,content):
    grep_process = Popen(["touch",filename], stdout=PIPE, stderr=PIPE)
    add_process = Popen(["echo", "",content,"", ">>",filename], stdin=grep_process.stdout, stdout=PIPE, stderr=PIPE)
    add_process.wait()
    return True if filename in get_all_files() else False

def remove_file(filename):
    remove_process = Popen(["rm",filename], stdout=PIPE, stderr=PIPE)
    remove_process.wait()
    return False if filename in get_all_files() else True

$ nano test_files.py

import pytest
import json
import unittest
import tempfile
import os
from files import app

class FlaskTestCase(unittest.TestCase):

    def test_create_file(self):
        client = app.test_client(self)
        result = client.post('/v1.0/files',data=json.dumps(dict(filename='new file',content='content')),follow_redirects=True)

```

```

        self.assertEqual(result.status_code,201)

def test_read_files(self):
    client = app.test_client(self)
    result = client.get('/v1.0/files',follow_redirects=True)
    self.assertEqual(result.status_code,200)

def test_delete_files(self):
    client = app.test_client(self)
    result = client.delete('/v1.0/files',follow_redirects=True)
    self.assertEqual(result.status_code,200)

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

$ nano recently_created.py

from flask import Flask, abort, request
from subprocess import Popen, PIPE
import json

app = Flask(__name__)
api_url = '/v1.0/files'

@app.route(api_url+' /recently_created',methods=['POST'])
def post_recently_created():
    return "NOT FOUND", 404

@app.route(api_url+' /recently_created',methods=['GET'])
def get_files():
    list = {}
    files = Popen(["find","-cmin","+0","-cmin","-180"], stdout=PIPE, stderr=PIPE)
    recent_files = Popen(["awk","-F','','',{print $2}"], stdin=files.stdout, stdout=PIPE, stderr=PIPE).communicate()[0].split('\n')
    list["files"] = filter(None,recent_files)
    return json.dumps(list), 200

@app.route(api_url+' /recently_created',methods=['PUT'])
def put_recently_created():
    return "NOT FOUND", 404

@app.route(api_url+' /recently_created',methods=['DELETE'])
def delete_recently_created():
    return "NOT FOUND", 404

if __name__ == "__main__":
    app.run(host='0.0.0.0',port=9090,debug='True')

$ nano test_recently_created.py

import pytest

```

```

import json
import unittest
import tempfile
import os
from files import app

class FlaskTestCase2(unittest.TestCase):

    def test_get_files(self):
        client = app.test_client(self)
        result = client.get('/v1.0/files/recently_created', follow_redirects=True)
        self.assertEqual(result.status_code, 200)

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

```

6. Ahora cree los archivos que permitirán ejecutar las pruebas unitarias con jenkins

```
$ nano run_tests.sh
```

```

#!/usr/bin/env bash
set -e

. ~/.virtualenvs/myproject/bin/activate

PYTHONPATH=. py.test --junitxml=python_tests_myproject.xml

$ nano run_tests_with_coverage.sh

#!/usr/bin/env bash
set -e

. ~/.virtualenvs/myproject/bin/activate

PYTHONPATH=. py.test --junitxml=python_tests_myproject.xml
PYTHONPATH=. py.test --cov-report xml --cov=../myproject
PYTHONPATH=. py.test --cov-report html --cov=../myproject

```

7. Habilite el servicio jenkins para que pueda operar en el puerto 8080 desde el usuario root

```

$ # chkconfig jenkins on
# service jenkins start
# iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT
# service iptables save

```

8. Cree y active un ambiente virtual en el usuario jenkins

su jenkins

```
$ mkdir /var/lib/jenkins/.virtualenvs $ cd /var/lib/jenkins/.virtualenvs $ virtualenv myproject $ . myproject/bin/activate
```


Cuando quiera desactivar el ambiente virtual ingrese

```
$ deactivate
```

9. Teniendo el ambiente activo, instale los requerimientos para la ejecución

```
$ pip install xmlrunner $ pip install unittest2 $ pip install pytest $ pip install pytest-cov $ pip install flask
```

10. Si desea probar su código por consola antes de usar jenkins, puede hacerlo por medio de pytest así:

```
 $ pip install pytest-cov  
$ pip install pytest-xdist  
$ py.test
```

11. Ahora configure su nuevo proyecto con nombre myproject en jenkins, no olvide relacionarlo con su repositorio de github donde está el código del proyecto y agregar las ejecuciones de coverage,html y junit. Además configure la ejecución del workspace con el archivo run_tests_with_coverage.sh, tal y como está en el video tutorial de la primer parte. Acceda por medio de la dirección 192.168.56.101:8080
12. Cuando haya creado su proyecto myproject, en la barra de opciones de la izquierda dé clic en construir ahora y en el número que se refleja con una punto de color puede ver el estado de la ejecución y la información relativa a ella. Si este punto es rojo es porque hubo errores y si es azul es porque se ejecutó correctamente.

Enlace Repositorio Github: <https://github.com/paulaandrea27/Parcial-2-Operativos.git>
(<https://github.com/paulaandrea27/Parcial-2-Operativos.git>)