Universidad Icesi

Christian Cárdenas

A00212740

Sistemas Operativos

Prof. Daniel Barragán

Repositorio: http://github.com/cdcardenas/so-exam2/

Parcial 2: Flask, Swagger y sqlite

1. El script background.py se encarga de ejecutar los comandos correspondientes para capturar la información de uso de memoria **RAM, CPU, disco duro** y estado del servicio **httpd,** como el requerimiento pide que este script se ejecute en segundo plano cada 60 segundos, se utilizó **crontab** para solucionar el problema.

crontab es un servicio del sistema operativo que permite agendar la ejecución de trabajos o jobs por usuario en este caso se usó el comando:

$ crontab -u check_user -e

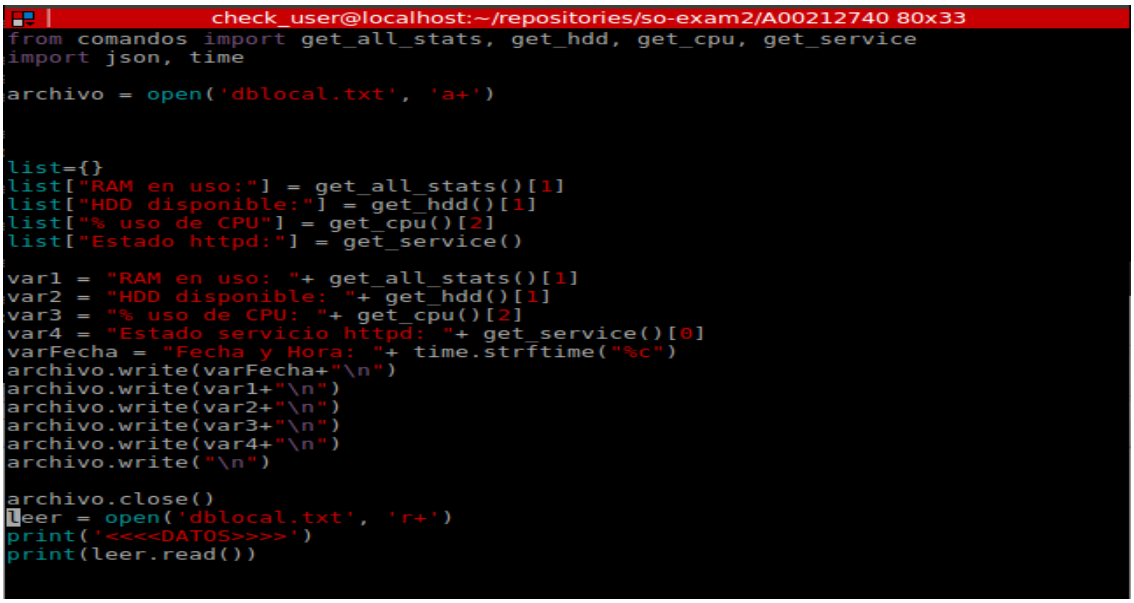
este comando permite editar el archivo de configuración de crontab en donde introducimos la línea:

* * * * * python ~/repositories/so-exam2/A00212740/background.py > ~/repositories/so-exam2/A00212740/dblocal.txt
* * * * * python ~/repositories/so-exam2/A00212740/consultas.py

La primera línea le indica a crontab que queremos que se ejecute cada él script background.py cada 60 segundo y que la salida que está generando el script sea almacenada en el archivo de texto dblocal.txt

La segunda línea ejecuta el script consultas.py cada minuto y guarda por medio de SQLite en la base de datos test.db

código fuente del script background.py:



```
* * * * * python ~/repositories/so-exam2/A00212740/background.py > ~/repositorie
s/so-exam2/A00212740/dblocal.txt
[root@localhost ~]#
```

1.2 código fuente del script consultas.py y del modelo de base de datos llamado modelo.py

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:////home/check_user/repositories/so-exam2/A00212740/test.db'
db = SQLAlchemy(app)


class Stats(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    cpu = db.Column(db.String(80), nullable=False)
    ram = db.Column(db.String(120), nullable=False)
    hdd = db.Column(db.String(120), nullable=False)
    service = db.Column(db.String(120), nullable=False)


    def __init__(self, cpu, ram, hdd, service):
        self.cpu = cpu
        self.ram = ram
        self.hdd = hdd
        self.service = service

    def __repr__(self):
        stats = self.cpu +" " + self.ram + " " + self.hdd+ " " + self.service

        return '<stats %r>' % stats
```

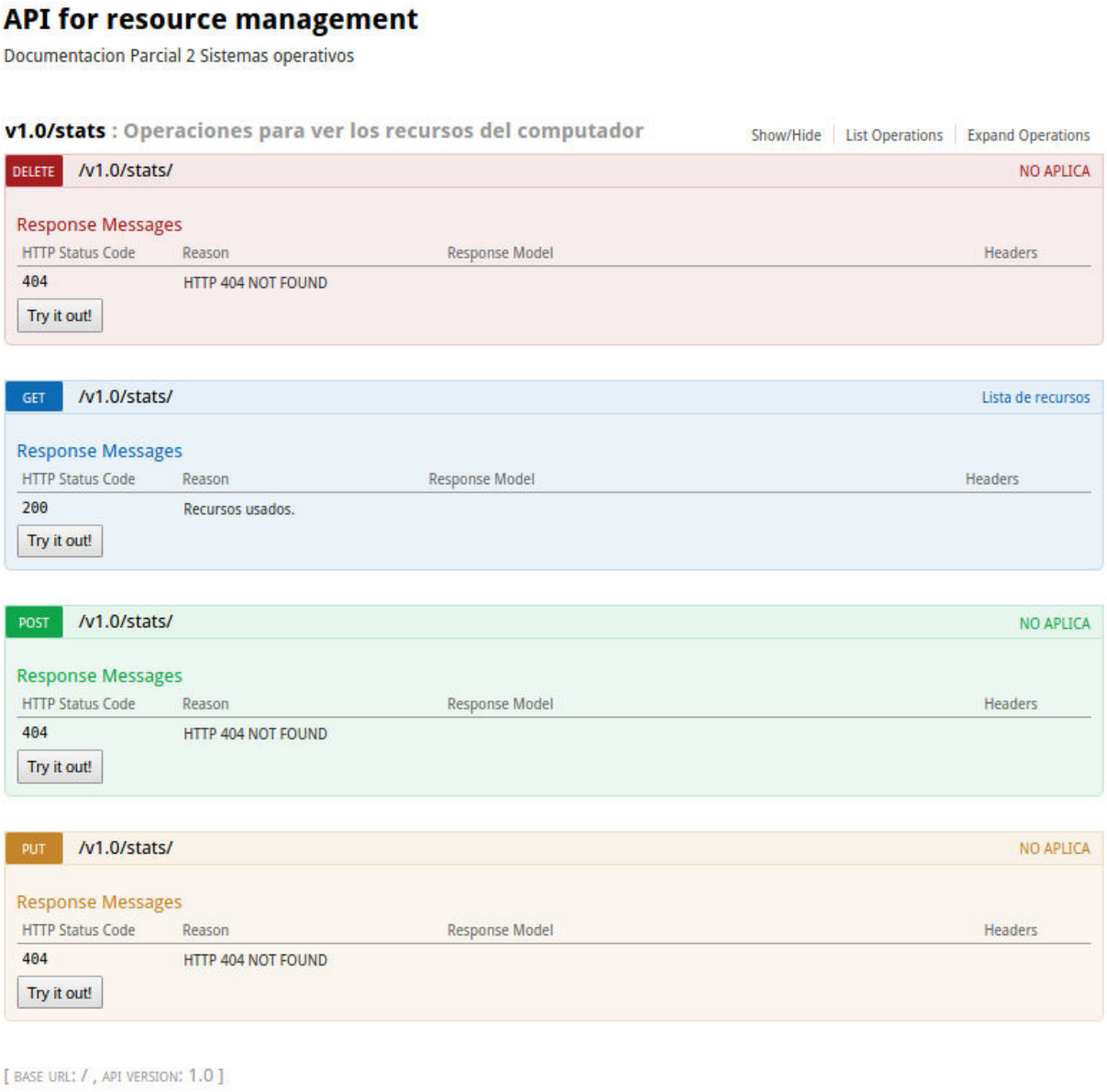
```
from modelo import db
from modelo import Stats
from comandos import get_all_stats, get_cpu, get_service, get_hdd



db.create_all()
#se capturan los datos según el formato indicado en modelo.py
stats = Stats(get cpu()[2], get_all_stats()[1], get_hdd()[1], get_service() )


#se agregan los cambios en la base de datos
db.session.add(stats)
#se guardan los cambios en la base de datos
db.session.commit()

#stats2 = Stats.query.all()
#print(stats2)
```

2. Para el despliegue del API se utilizó Flask con el siguiente código fuente:

```
from flask import Flask, abort, request
import json
from flask_restplus import Resource, Api
from flask_restplus import fields

from comandos import get_all_stats, get_hdd, get_cpu, get_service

app = Flask(__name__)
#api_url= '/v1.0'
api = Api(app,version='1.0', title='API for resource management', description=
'Documentacion Parcial 2 Sistemas operativos')

ns = api.namespace('v1.0/stats', description='Operaciones para ver los recurso
s del computador')
@ns.route('/')
#@app.route(api_url+'/stats', methods=['GET'])
class StatsCollection(Resource):
 @api.response(200, 'Recursos usados.')
 def get(self):
  """Lista de recursos"""
  list = {}
  list["RAM en uso:"] = get_all_stats()[1]
  list["HDD disponible:"] = get_hdd()[1]
  list["% uso de CPU"] = get_cpu()[2]
  list["Estado httpd:"] = get_service()
  return json.dumps(list),200

 @api.response(404, 'HTTP 404 NOT FOUND')
 def post(self):
  """NO APLICA"""
  return "HTTP 404 NOT FOUND", 404

 @api.response(404, 'HTTP 404 NOT FOUND')
 def put(self):
  """NO APLICA"""
  return "HTTP 404 NOT FOUND", 404

 @api.response(404, 'HTTP 404 NOT FOUND')
 def delete(self):
  """NO APLICA"""
  return "HTTP 404 NOT FOUND", 404

if __name__=="__main__":

  app.run(host='0.0.0.0', port=8080, debug='True')
~
~
```

Esta API permite mediante una petición tipo GET mostrarle al usuario el consumo de recursos del equipo. La documentación de la API se realizó con flask restplus y SwaggerUI que permiten generar documentación de forma ordenada.

A la documentación se puede acceder mediante la url: Dirección_IP:Puerto



GET /v1.0/stats/ Lista de recursos

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Recursos usados.		

Try it out! Hide Response

Curl

```
curl -X GET --header 'Accept: application/json' 'http://192.168.57.2:8080/v1.0/stats/'
```

Request URL

```
http://192.168.57.2:8080/v1.0/stats/
```

Response Body

```
"{\"HDD disponible:\": \"16G\", \"% uso de CPU\": \"0,00\", \"Estado httpd:\": [\" inactive (dead)\"], \"RAM en uso:\":
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Wed, 03 May 2017 19:09:17 GMT",
  "server": "Werkzeug/0.12.1 Python/2.7.5",
  "content-length": "154",
  "content-type": "application/json"
}
```

A continuación, se muestra la salida de la petición GET en el explorador de internet donde se puede visualizar el consumo de recursos de la CPU.

192.168.57.2:8080/v1.0/stats/

"{\"HDD disponible:\": \"16G\", \"% uso de CPU\": \"0,00\", \"Estado httpd:\": [\" inactive (dead)\"], \"RAM en uso:\": \" 160 m used memory\"}"

Para los comandos usados para obtener la información del sistema fueron:

RAM: vmstat -s -S m
HDD: df /dev/mapper/cl-root -h awk {print $4}
CPU: sar 1 1 | awk {print $5}
Servicio: /usr/sbin/service httpd status | awk -F ‘Active:’ {print $2}

```
from subprocess import Popen, PIPE

def get_all_stats():
  grep_process = Popen(["vmstat", "-s","-S","m"], stdout=PIPE,stderr=PIPE)
  listado_stats= Popen(["awk", '-F', ':','{print $1}' ],stdin=grep_process.std
out, stdout=PIPE, stderr=PIPE).communicate()[0].split('\n')

  print(listado_stats)
  return filter(None, listado_stats)


def get_hdd():
  grep_process = Popen(["df", "/dev/mapper/cl-root", "-h"], stdout=PIPE, stder
r=PIPE)
  lista= Popen(["awk",'{print $4}' ],stdin=grep_process.stdout, stdout=PIPE, s
tderr=PIPE).communicate()[0].split('\n')
  return filter(None, lista)

def get_cpu():
  grep_process = Popen(["sar", "1", "1"], stdout=PIPE, stderr=PIPE)
  lista = Popen(["awk", '{print $5}'], stdin=grep_process.stdout, stdout=PIPE,
 stderr=PIPE).communicate()[0].split('\n')
  return filter(None, lista)

def get_service():
  grep_process = Popen(["/usr/sbin/service", "httpd", "status"], stdout=PIPE,
stderr=PIPE)
  lista = Popen(["awk", '-F', 'Active:', '{print $2}'], stdin=grep_process.std
out, stdout=PIPE, stderr=PIPE).communicate()[0].split('\n')

  return filter(None, lista)
```