Relatório do Trabalho Prático 3

Replicação de Serviços Web

Identificação

Pedro Antônio de Souza (201810557)

Descrição Geral da Solução

A ferramenta NGINX é um servidor web de alto desempenho que, originalmente, tem como foco a eficiência no gerenciamento de concorrência utilizando poucos recursos. O NGINX é arquitetado como um proxy reverso. Sendo assim, é possível utilizá-lo como balanceador de cargas.

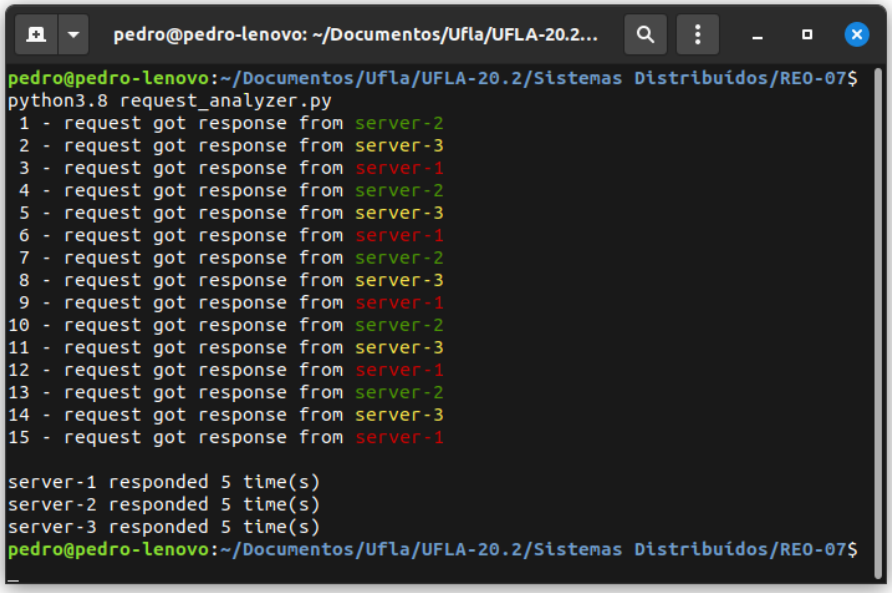
As configurações do NGINX permitem balancear cargas através de vários mecanismos. O mecanismo de ip-hash sempre direciona um cliente, baseado em seu IP, para um mesmo servidor. Essa técnica é útil, por exemplo, quando pretende-se persistir uma sessão de um cliente em uma aplicação no servidor.

Para executar o balanceamento de cargas, três servidores HTTP foram implementados utilizando Javascript simulando um serviço replicado. A implementação foi baseada no código disponibilizado na sala virtual da disciplina, diferindo-se apenas no comando de log inserido para indicar o recebimento de requisições (veja os códigos no anexo ao final deste documento). Por fim, o NGINX foi configurado para balancear a carga entre esses três servidores.

Um programa em Python foi desenvolvido para simular uma aplicação cliente efetuando as requisições diretamente para o endereço do servidor principal. O cliente envia 15 requisições e imprime no terminal a origem da resposta obtida. Também é indicada a quantidade de vezes que cada um dos três servidores responderam às requisições.

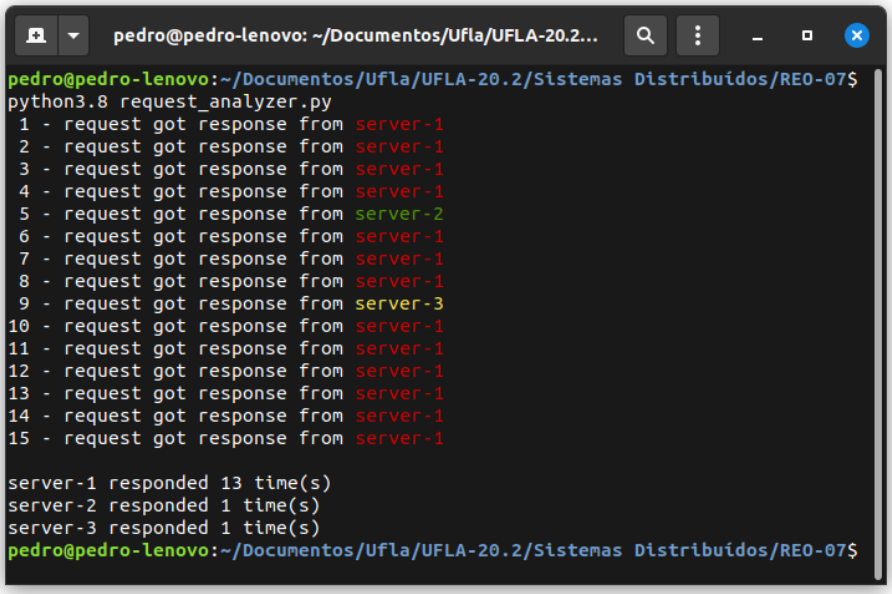
Resultados

O resultado da execução da aplicação cliente, que efetuou 15 requisições para o servidor web, mostra que cada uma das réplicas respondeu a cinco requisições. A imagem abaixo ilustra essa execução.



Esses números indicam que o balanceamento foi realizado com sucesso pelo NGINX. As imagens abaixo mostram a execução do servidor 1, 2 e 3, respectivamente.

Os testes ilustrados acima ocorreram com os três servidores possuindo o mesmo peso nas configurações do NGINX. Para verificar outras possibilidades de balanceamento, foi definido peso 10 para o servidor 1 (porta 3001). Os resultados das requisições podem ser observados na figura abaixo.



Dificuldades

O objetivo dessa atividade foi simular o balanceamento de carga em servidores web. Porém, não foi possível realizar simulações de alta fidelidade já que os servidores e o cliente implementados para essa simulação estão na mesma máquina. Com uma latência desprezível e 100% de disponibilidade, situações de falha de requisições não puderam ser experimentadas.

ANEXO A - Código-fonte do cliente

request_analyzer.py

```
import requests

def print_count(server_response_count: dict):
    for server, count in server_response_count.items():
        print(f'{server} responded {count} time(s)')

def print_response(request_index: int, server_identifier: str):
    server_color = {'server-1': '\033[31m', 'server-2': '\033[32m',
'server-3': '\033[93m', 'end': '\033[0m'}
    color = server_color[server_identifier]
    end_color = server_color['end'];
    print(f'{request_index:>2} - request got response from
{color}{server_identifier}{end_color}')

def send_requests(number_of_requests: int):
    server_response_count = {'server-1': 0, 'server-2': 0, 'server-3': 0}

    for i in range(1, number_of_requests + 1):
        try:
            response = requests.get('http://localhost:8080/')
            server_identifier = response.text;
            server_response_count[server_identifier] += 1

            print_response(i, server_identifier)
        except:
            print(f'{i:>2} - can\'t reach the server')

    return server_response_count

def main():
    server_response_count = send_requests(15)
    print()
    print_count(server_response_count)

if __name__ == "__main__":
    main()
```

ANEXO B - Código-fonte do servidor 1

server-1.js

```
const http = require('http');

const server = http.createServer((req, res) => {
    console.log('receiving request');

    res.setHeader("Content-Type", "application/json");
    res.writeHead(200);
    res.end("server-1");
});

server.listen(3001, 'localhost', () => {
    console.log('running on http://localhost:3001/');
});
```

ANEXO C - Código-fonte do servidor 2

server-2.js

```
const http = require('http');

const server = http.createServer((req, res) => {
    console.log('receiving request');

    res.setHeader("Content-Type", "application/json");
    res.writeHead(200);
    res.end("server-2");
});

server.listen(3001, 'localhost', () => {
    console.log('running on http://localhost:3002/');
});
```

ANEXO D - Código-fonte do servidor 3

server-3.js

```
const http = require('http');

const server = http.createServer((req, res) => {
    console.log('receiving request');

    res.setHeader("Content-Type", "application/json");
    res.writeHead(200);
    res.end("server-3");
});

server.listen(3001, 'localhost', () => {
    console.log('running on http://localhost:3003/');
});
```