UNIVERSIDADE PAULISTA

APS - SISTEMA DE WEB SITES COM MÍNIMO DE TRIPLA REDUNDÂNCIA

RELATÓRIO SOBRE O DESENVOLVIMENTO DE UM SISTEMA

DE WEB SITES UTILIZANDO DOCKER

DANIEL LIBERATO - RA T0740B-4 | TURMA: CC8Q12

FELIPE SCHERER - RA D88HJE-1 | TURMA: CC8P12

JEHAN MARQUES MENDONÇA DIAS - RA N465CC-1 | TURMA: CC8P12

KELLY DENA - RA D912JB-8 | TURMA: CC8P12

VINÍCIUS GUIDI ROSSI – RA N386BH-8 | TURMA: CC8P12

ORIENTADOR: GIOVANE MORAIS

CURSO: CIÊNCIA DA COMPUTAÇÃO - NOTURNO

CAMPINAS - SP

NOVEMBRO / 2022

SUMÁRIO

PROBLEMÁTICA

Deve ser criado um sistema de Web site com pelo menos redundância tripla na infraestrutura respeitando as seguintes regras:

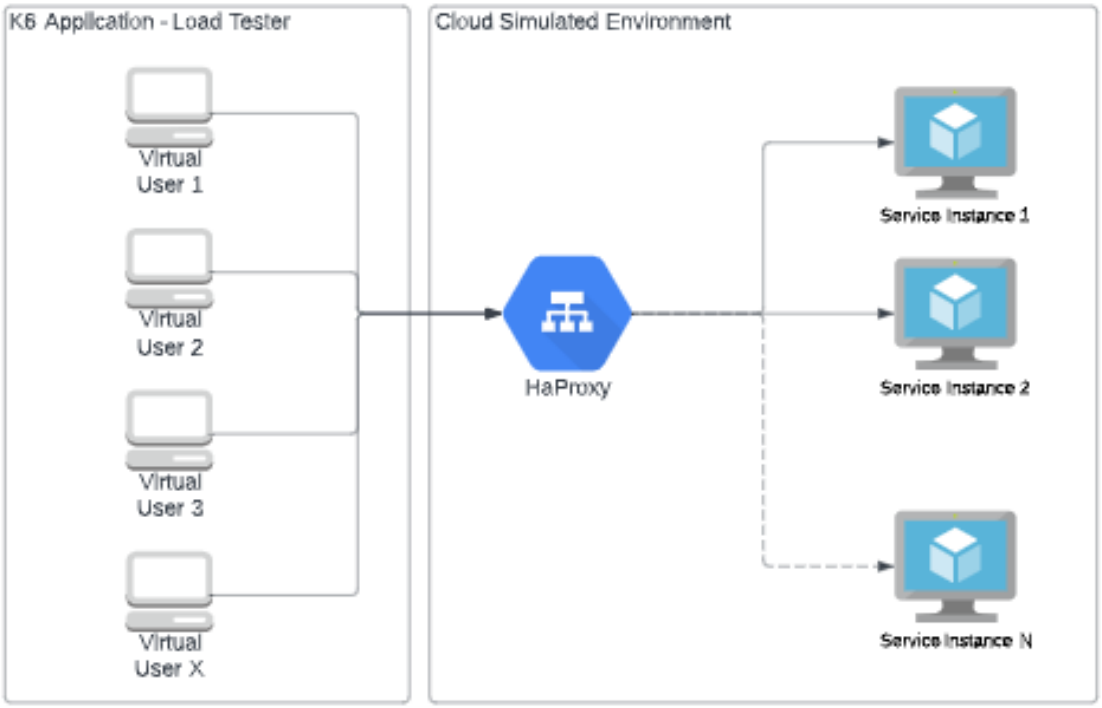
- O usuário deve acessar toda a infraestrutura por uma única e imutável URL;
- A requisição deve ser balanceada dentro do sistema de redundância tripla;
- O usuário não pode saber em qual servidor ou ambiente está;
- O conteúdo de cada servidor deve ser igual;
- O sistema como um todo deve ser tolerante a falhas. Se um ou mais ambientes caírem, os acessos do usuário devem ser redirecionados automaticamente e de forma transparente.

PROPOSTA DE SOLUÇÃO

Utilizar uma aplicação do tipo *Load Balancer* (Balanceador de carga) para redirecionar as chamadas para várias instâncias da mesma aplicação.

Essa aplicação receberá todas as requisições HTTP enviadas para um determinado IP (ou domínio) e irá distribuir essas chamadas para as instâncias da aplicação que estiverem disponíveis. Caso algumas das aplicações não estiverem disponíveis, o Balanceador de Carga irá perceber este cenário através da técnica de *health-check*, e, somente então, não irá repassar as chamadas recebidas para aquela determinada instância.

O *health-check* consiste em, de tempos em tempos, chamar a aplicação que deseja-se verificar a disponibilidade. Muitas vezes essa verificação é feita através da camada 3 (camada de Rede no modelo OSI) utilizando o *Internet Control Message Protocol* (ICMP). Porém, há outras alternativas como definir um *endpoint* específico que irá retornar uma resposta estática caso a aplicação esteja funcionando corretamente.



Nossa infraestrutura se dará por um *Load Balancer* que receberá as chamadas dos usuários (*Virtual User*) e irá repassar (*proxy*) essas chamadas para uma lista de serviços (*Service Instance*). Estes serviços irão responder uma mensagem fixa ao usuário. Essa mensagem será no formato JSON e terá o conteúdo “API #X”, onde X é o identificador daquele serviço. Como cada serviço terá seu próprio identificador, o conteúdo da resposta será diferente para cada serviço. Isso foi propositalmente pensado para podermos identificar que os serviços estão recebendo as chamadas repassadas pelo HaProxy e também saber qual está respondendo àquela chamada.

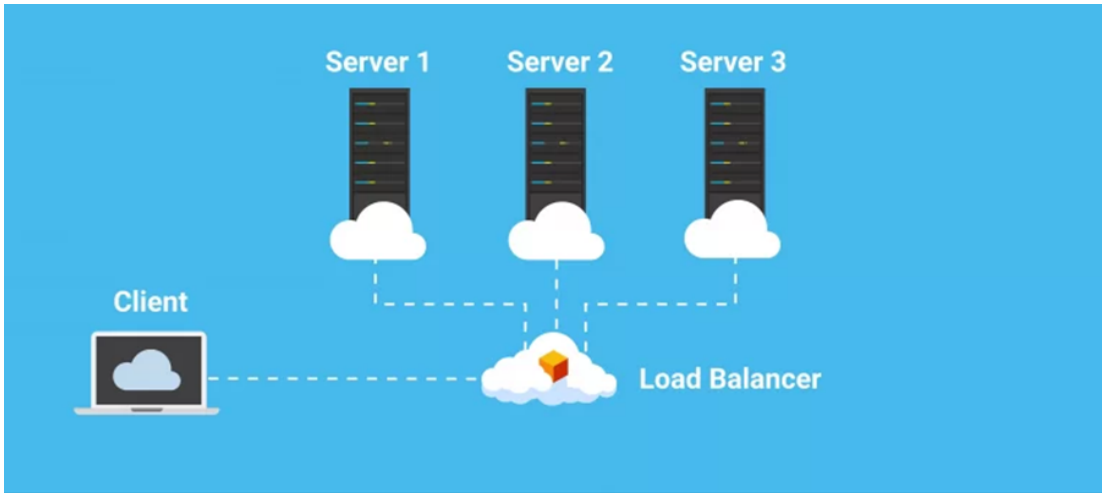
FERRAMENTAS UTILIZADAS

A ferramenta utilizada para efetuar o balanceamento de carga do sistema foi o HaProxy, já para efetuar a redundância do sistema foi utilizado o Docker com Docker-compose. Todo o sistema de infraestrutura foi instalado em um sistema operacional Linux em local hosts. Ademais, todos os testes de resiliência, adição de recursos e remoção de recursos foram realizados com o auxílio da ferramenta K6.

HaProxy e Auto Scaling

Proxy utiliza a técnica Load Balance para que o tráfego de requisições seja distribuído de forma equilibrada entre os contêineres utilizados no Docker. Assim a carga individual de cada container é dividida e garante que cada usuário tenha um acesso a aplicação com um melhor desempenho, um tempo de resposta mais baixo e menos sobrecarregar entre os containers do Docker.

Toda vez que o sistema receber uma requisição o HaProxy irá direcionar esta requisição para algum container que esteja disponível no momento, o direcionamento acontece de forma aleatória e automática, sem que o usuário perceba. Além disso, o HaProxy consegue identificar se um container está ou não ativo, caso haja algum problema com o contêiner o usuário será redirecionado para outro contêiner que esteja funcional.



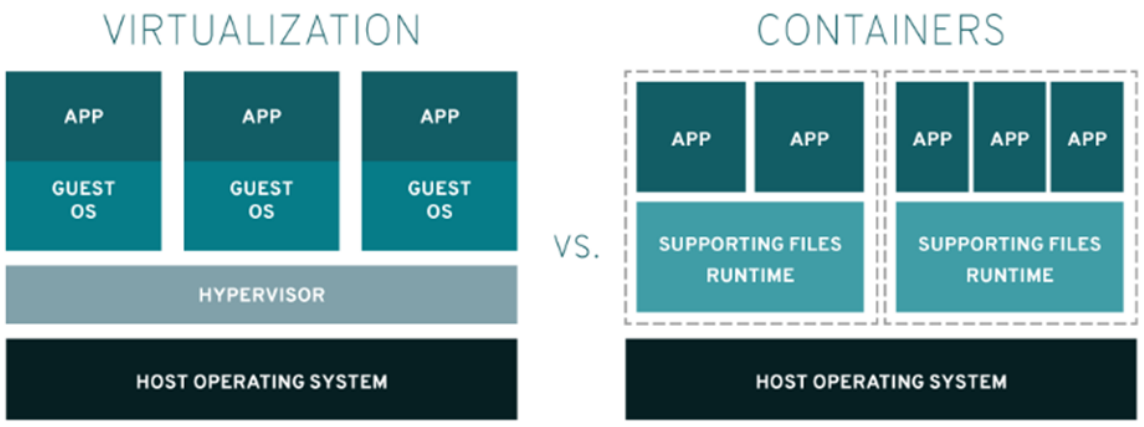
Fonte: https://www.hostgator.com.br/blog/load-balance-o-que-e-e-como-implementar/

Outro recurso utilizado foi o Auto Scaling, porém para utilizar este conceito é necessário possuir uma ferramenta de load balance, no caso deste trabalho a ferramenta utilizada é o HaProxy. O Auto Scaling permite que o sistema consiga aumentar ou diminuir a quantidade de recursos de forma automática dependendo da quantidade de requisições que o sistema está recebendo em um determinado momento. Por isso o HaProxy é de suma importância, se a quantidade de requisições aumentar, o Auto Scaling vai aumentar os recursos de forma horizontal e automática e o HaProxy vai distribuir as requisições entre esses novos recursos, e o mesmo acontece quando as requisições diminuem, o Auto Scaling vai diminuir os recursos e o HaProxy vai redirecionar os usuários entre os containers ativos.

Docker

O Docker é uma plataforma open source que possibilita criar diversas aplicações em uma mesma máquina. O Docker permite que a aplicação seja empacotada dentro de um container, com isso, a aplicação fica isolada das demais sem que haja conflito entre as aplicações.

O Docker tem um funcionamento parecido com uma máquina virtual, porém a grande diferença está no desempenho. Em um sistema de virtualização convencional é necessário utilizar um sistema operacional para cada aplicação, já o Docker utiliza contêiner para rodar cada aplicação. Esses containers fornecem um ambiente totalmente isolado e eles possuem somente o necessário para rodar uma aplicação específica.



Fonte: https://www.redhat.com/pt-br/topics/virtualization

Docker Compose

Docker Compose é uma ferramenta para rodar múltiplos containers Docker de uma forma fácil e rápida. Esta ferramenta utiliza um arquivo do tipo YML que é utilizado para referenciar as imagens Docker e inicializar os containers de acordo com os comandos e configurações descritos neste arquivo.

No nosso projeto, o Docker compose é utilizado da seguinte forma: cinco das nossas instâncias da nossa aplicação são iniciadas além do HaProxy. Utilizamos uma configuração do Docker Compose que se chama “depends-on”, onde condicionamos a execução de um container à inicialização de outro, dessa forma inicializamos todas as nossas aplicações primeiro e, somente então, inicializamos o Load Balancer.

K6

A ferramenta K6 é amplamente utilizada para realizar testes de desempenho, esta ferramenta é gratuita e de código aberto. Com esta ferramenta é possível colar uma determinada carga no servidor e ver como o servidor irá se comportar. O k6 é desenvolvido em JavaScript, portanto ele é uma ferramenta de alto desempenho projetado para realizar testes de alta carga.

Em nosso projeto, a ferramenta foi utilizada para verificar se o serviço está dividindo a carga entre os containers e verificar também se a redundância do sistema está funcionando corretamente.

Shell Script

Shell Script é um arquivo que será interpretado por algum programa do tipo Shell, como *sh*, *bash*, *zsh*, etc.

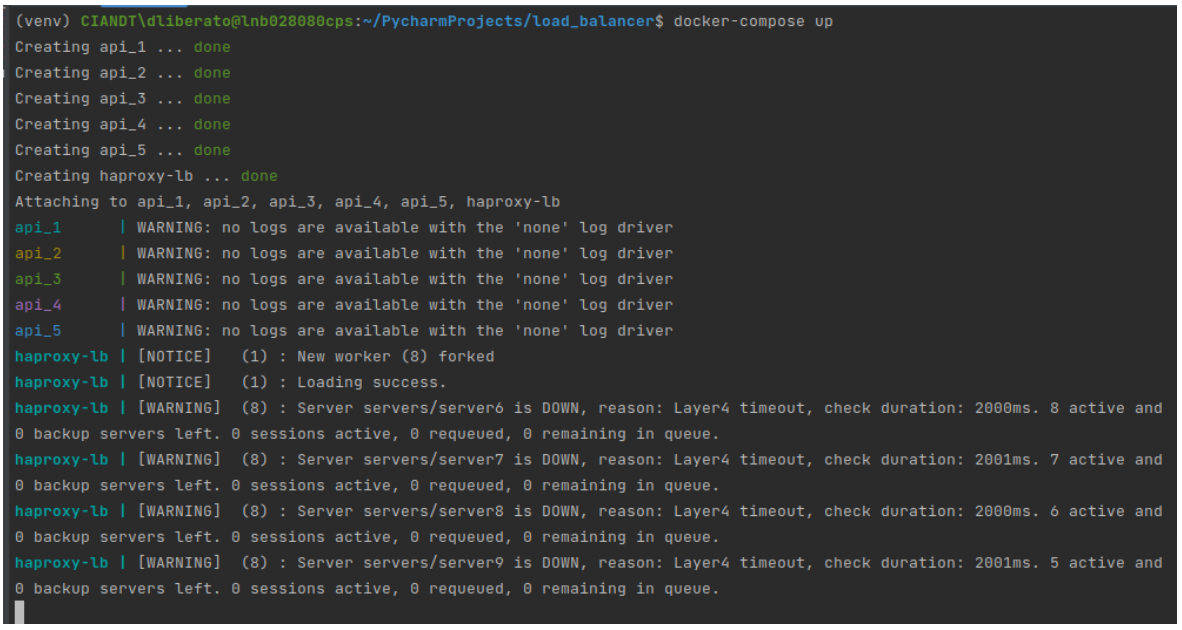
Em nosso projeto, utilizamos o Shell Script apenas para facilitar as execuções dos comandos Docker. Em outras palavras, criamos um arquivo Shell que aceita um comando e um número. Esses parâmetros representam a ação que queremos dar a um de nossos containers (iniciar ou parar) e o identificador deste container, respectivamente.

TESTE DE RESILIÊNCIA E REMOÇÃO DE RECURSOS

- *Inicialização da infraestrutura*

```
docker-compose up
```

A execução do comando acima irá resultar na inicialização dos containers de nossas APIs e também do container do HaProxy. Podemos notar isso através das mensagens de “Creating api_1 … Done” por exemplo, e também pela mensagem do container de nome “haproxy-lb” em azul. É possível ver também que o HaProxy exibiu mensagens dizendo que os containers de 6 a 9 estão *offline*. Ou seja, o *Load Balancer* identificou que as aplicações de 1 a 5 estão online.



A seguinte imagem mostra a execução do comando que inicia nossa simulação de usuários. Na área interna da imagem, temos os logs do K6. Esses logs estão no seguinte formato: “INFO[<segundo>] API#<identificador da API>”. Portanto podemos ver que, inicialmente, as APIs de 1 a 5 estão respondendo corretamente.

```
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$ k6 run k6/load_test.js

         /\      |‾‾| /‾‾/   /‾‾/
    /\  /  \     |  |/  /   /  /
   /  \/    \    |     (   /   ‾‾\
  /          \   |  |\  \ |  (‾)  |
 / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: k6/load_test.js
     output: -

  scenarios: (100.00%) 1 scenario, 1 max VUs, 5m30s max duration (incl. graceful stop):
           * default: Up to 1 looping VUs for 5m0s over 1 stages (gracefulRampDown: 30s, gracefulStop: 30s)

INFO[0000] API #4                                        source=console
INFO[0001] API #5                                        source=console
INFO[0002] API #1                                        source=console
INFO[0003] API #2                                        source=console
INFO[0004] API #3                                        source=console
INFO[0005] API #4                                        source=console
INFO[0006] API #5                                        source=console
INFO[0007] API #1                                        source=console
running (0m07.2s), 1/1 VUs, 7 complete and 0 interrupted iterations
default   [--------------------------------------] 1/1 VUs  0m07.2s/5m00.0s
```

- *Derrubar parte da infra e mostrar que ela continua acessível*

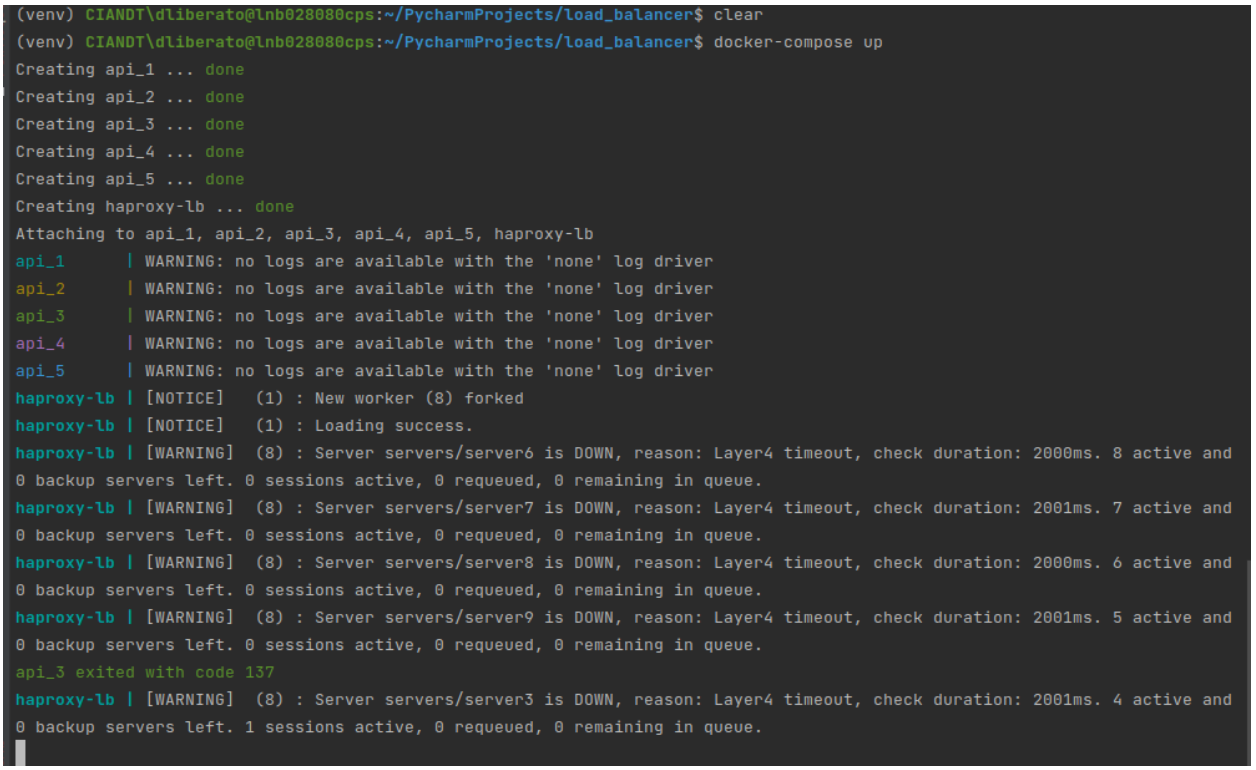
O seguinte comando irá finalizar a execução do container onde a API número 3 está inicializada:

```
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$ ./scripts/instance_commands.sh stop 3
api_3
api_3
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$
```

Na seguinte imagem podemos notar que os usuários não receberam nenhum erro enquanto acessavam nossa infraestrutura:

```
INFO[0022] API #1                                        source=console
INFO[0023] API #2                                        source=console
INFO[0024] API #3                                        source=console
INFO[0025] API #4                                        source=console
INFO[0026] API #5                                        source=console
INFO[0027] API #1                                        source=console
INFO[0028] API #2                                        source=console
INFO[0029] API #3                                        source=console
INFO[0030] API #4                                        source=console
INFO[0031] API #5                                        source=console
INFO[0032] API #1                                        source=console
INFO[0033] API #2                                        source=console
INFO[0054] API #4                                        source=console
INFO[0055] API #5                                        source=console
INFO[0056] API #1                                        source=console
INFO[0057] API #2                                        source=console
INFO[0058] API #4                                        source=console
INFO[0059] API #5                                        source=console
INFO[0060] API #1                                        source=console
INFO[0061] API #2                                        source=console
INFO[0062] API #4                                        source=console
INFO[0063] API #5                                        source=console
INFO[0064] API #1                                        source=console
INFO[0065] API #2                                        source=console
INFO[0066] API #4                                        source=console
INFO[0067] API #5                                        source=console
INFO[0068] API #1                                        source=console
```

Na seguinte imagem podemos notar dois pontos importantes: a linha verde com a mensagem “**api_3 exited with code 137**” indica que a API 3 teve sua execução interrompida; E, logo em seguida temos a mensagem do HaProxy confirmando que a API 3 não está mais *online*:



TESTE DE ADIÇÃO DE RECURSOS

- *Subir mais ambientes de forma transparente*

Adição de recurso (Inicializando a API 8):

```
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$ ./scripts/instance_commands.sh start 8
Sending build context to Docker daemon  90.73MB
Step 1/13 : FROM python:3.9.4-slim AS build
 ---> 6b6ba86a5f35
Step 2/13 : WORKDIR /app
 ---> Using cache
 ---> 99d218dc6227
Step 3/13 : ENV PYTHONDONTWRITEBYTECODE 1
 ---> Using cache
 ---> 748735995989
Step 4/13 : ENV PYTHONUNBUFFERED 1
 ---> Using cache
 ---> 8623f98dd24f
Step 5/13 : ARG app_id
 ---> Using cache
 ---> 2dd36a1adbf8
Step 6/13 : RUN echo "app_id = $app_id"
 ---> Using cache
 ---> 3fa83254be3b
Step 7/13 : ENV APPLICATION_ID $app_id
 ---> Using cache
 ---> 62425fb5727d
Step 8/13 : COPY requirements.txt .
 ---> Using cache
 ---> f96706bbe949
Step 9/13 : RUN pip install --upgrade pip
 ---> Using cache
 ---> 147a7b2b044f
Step 10/13 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Using cache
 ---> 1bf772d969eb
Step 11/13 : COPY src /app
 ---> Using cache
 ---> 6290de38f515
Step 12/13 : EXPOSE 8080
 ---> Using cache
 ---> 8d7ef748674e
Step 13/13 : CMD python -m uvicorn main:app --reload --host 0.0.0.0 --port 8080
 ---> Using cache
 ---> 71f100c1ae78
Successfully built 71f100c1ae78
Successfully tagged api_8:latest
6d01d918e24195224a74e263f8f5f0bde794ab2c242209e5b5697e47d74ee3f2
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$
```

Adição de recurso (API 8 começa a responder aos usuários no segundo #0166):

```
INFO[0139] API #5                                source=console
INFO[0140] API #1                                source=console
INFO[0141] API #2                                source=console
INFO[0142] API #4                                source=console
INFO[0143] API #5                                source=console
INFO[0144] API #1                                source=console
INFO[0145] API #2                                source=console
INFO[0146] API #4                                source=console
INFO[0147] API #5                                source=console
INFO[0148] API #1                                source=console
INFO[0149] API #2                                source=console
INFO[0150] API #4                                source=console
INFO[0151] API #5                                source=console
INFO[0152] API #1                                source=console
INFO[0153] API #2                                source=console
INFO[0154] API #4                                source=console
INFO[0155] API #5                                source=console
INFO[0156] API #1                                source=console
INFO[0157] API #2                                source=console
INFO[0158] API #4                                source=console
INFO[0159] API #5                                source=console
INFO[0160] API #1                                source=console
INFO[0161] API #2                                source=console
INFO[0162] API #4                                source=console
INFO[0163] API #5                                source=console
INFO[0164] API #1                                source=console
INFO[0165] API #2                                source=console
INFO[0166] API #8                                source=console
INFO[0167] API #4                                source=console
INFO[0168] API #5                                source=console
INFO[0169] API #1                                source=console
INFO[0170] API #2                                source=console
INFO[0171] API #8                                source=console
INFO[0172] API #4                                source=console
INFO[0173] API #5                                source=console
INFO[0174] API #1                                source=console
INFO[0175] API #2                                source=console
INFO[0176] API #8                                source=console
INFO[0177] API #4                                source=console
INFO[0178] API #5                                source=console
INFO[0179] API #1                                source=console
INFO[0180] API #2                                source=console
INFO[0181] API #8                                source=console
INFO[0182] API #4                                source=console
```

Adição de recurso (HaProxy identificando que a API 8 está online):

```
(venv) CIANDT\dliberato@lnb028080cps:~/PycharmProjects/load_balancer$ docker-compose up
Creating api_1 ... done
Creating api_2 ... done
Creating api_3 ... done
Creating api_4 ... done
Creating api_5 ... done
Creating haproxy-lb ... done
Attaching to api_1, api_2, api_3, api_4, api_5, haproxy-lb
api_1       | WARNING: no logs are available with the 'none' log driver
api_2       | WARNING: no logs are available with the 'none' log driver
api_3       | WARNING: no logs are available with the 'none' log driver
api_4       | WARNING: no logs are available with the 'none' log driver
api_5       | WARNING: no logs are available with the 'none' log driver
haproxy-lb | [NOTICE]   (1) : New worker (8) forked
haproxy-lb | [NOTICE]   (1) : Loading success.
haproxy-lb | [WARNING]  (8) : Server servers/server6 is DOWN, reason: Layer4 timeout, check duration: 2000ms. 8 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
haproxy-lb | [WARNING]  (8) : Server servers/server7 is DOWN, reason: Layer4 timeout, check duration: 2001ms. 7 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
haproxy-lb | [WARNING]  (8) : Server servers/server8 is DOWN, reason: Layer4 timeout, check duration: 2000ms. 6 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
haproxy-lb | [WARNING]  (8) : Server servers/server9 is DOWN, reason: Layer4 timeout, check duration: 2001ms. 5 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
api_3 exited with code 137
haproxy-lb | [WARNING]  (8) : Server servers/server3 is DOWN, reason: Layer4 timeout, check duration: 2001ms. 4 active and
0 backup servers left. 1 sessions active, 0 requeued, 0 remaining in queue.
haproxy-lb | [WARNING]  (8) : Server servers/server8 is UP, reason: Layer4 check passed, check duration: 0ms. 5 active and
0 backup servers online. 0 sessions requeued, 0 total in queue.
```

CÓDIGO

haproxy/dockerfile

Comandos que o docker irá subir para o haproxy (Expor as portas 8084 e 80 para esse container).

haproxy > Dockerfile

```
FROM haproxy:latest

#RUN sudo chmod 777 /usr/local/etc/haproxy/haproxy.cfg

COPY haproxy/haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg

EXPOSE 8084
EXPOSE 80

RUN cat /usr/local/etc/haproxy/haproxy.cfg

```

haproxy/haproxy.cfg

Configuração do haproxy que irá subir para o docker.

haproxy > haproxy.cfg

```
# Simple configuration for an HTTP proxy listening on port 80 on all
# interfaces and forwarding requests to a single backend "servers" with a
# single server "server1" listening on 127.0.0.1:8000
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 30000ms
    timeout server 30000ms

frontend http-in
    bind *:80
    default_backend servers

frontend stats
    bind *:8084
    stats enable
    stats auth admin:admin
    stats uri /stats
    stats refresh 5s

backend servers
#    stats enable
#    stats auth admin:admin
#    stats uri /stats
    server server1 192.0.0.11:8080 check
    server server2 192.0.0.12:8080 check
    server server3 192.0.0.13:8080 check
    server server4 192.0.0.14:8080 check
    server server5 192.0.0.15:8080 check
    server server6 192.0.0.16:8080 check
    server server7 192.0.0.17:8080 check
    server server8 192.0.0.18:8080 check
    server server9 192.0.0.19:8080 check

```

k6/load_tests.js

Configuração do K6 para simular usuários ativos.

```
k6 > JS load_test.js > ...
1   import http from 'k6/http';
2   import { check, sleep } from 'k6';
3
4   export const options = {
5     stages: [
6       { duration: '5m', target: 1 },
7     ],
8   };
9
10  export default function () {
11      let res = http.get("http://0.0.0.0:80"); // local's LB
12    if(res.status == 200){
13      console.log(res.json('message'))
14    }else{
15      console.log("Error | status = " + res.status);
16    }
17    sleep(1);
18  }
19
```

scripts/instance_commands.sh

Script que sobe um container com a API e o id respectivo do container. Nesse caso, podemos criar o container o ID (*start*) ou parar o container específico (*stop*).

```
scripts > $ instance_commands.sh
1   #!/bin/bash
2
3   case "$1" in
4       start)
5         if [ $2 ];
6         then
7           docker build -t api_$(echo $2) --build-arg app_id=$(echo $2) .
8           docker run -d --name api_$(echo $2) --network load_balancer_public_net
            --ip 192.0.0.1$(echo $2) -p 808$(echo $2):8080 api_$(echo $2)
9         else
10          echo Error: Provide an service identification between 1 and 9 as
            argument.
11        fi
12      ;;
13
14      stop)
15        if [ $2 ];
16        then
17          docker stop api_$(echo $2)
18          docker rm api_$(echo $2)
19        else
20          echo Error: Provide an service identification between 1 and 9 as
            argument.
21        fi
22      ;;
23
24      *)
25        echo "Options are:"
26        echo "start {appId} | To create a new service instance (argument
          required)"
27        echo "stop {appId} | To stop a service instance (argument required)"
28        echo "Arguments are service IDs, and must be between 1 and 9"
29      ;;
30  esac
```

docker-compose.yml

Arquivo onde estão todas as configurações do docker para essa aplicação. Aqui temos as configurações dos serviços, portas e IPs.

docker-compose.yml

```yaml
# Taken from https://medium.com/swlh/beautiful-load-testing-with-k6-and-docker-compose-4454edb3a2e3
# This docker-compose file sets up a HaProxy and 3 simple APIs
version: '3.4'

services:
  api_1:
    container_name: api_1
    logging:
      driver: none
    build:
      context: .
      dockerfile: Dockerfile
      args:
        app_id: 1
    environment:
      - APPLICATION_ID=1
    networks:
      public_net:
        ipv4_address: 192.0.0.11
```

```yaml
  api_5:
    container_name: api_5
    logging:
      driver: none
    depends_on:
      - api_4
    build:
      context: .
      dockerfile: Dockerfile
      args:
        app_id: 5
    environment:
      - APPLICATION_ID=5
    networks:
      public_net:
        ipv4_address: 192.0.0.15

  haproxy:
    container_name: haproxy-lb
    depends_on:
      - api_5
    build:
      context: .
      dockerfile: haproxy/Dockerfile
    restart: always
    ports:
      - "80:80"
      - "8480:8480"
    networks:
      public_net:
        ipv4_address: 192.0.0.33

networks:
  public_net:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.0.0.0/24
```

dockerfile (root)

Arquivo do container que estará com a aplicação.

Dockerfile

```
# Dockerfile

# pull the official docker image
FROM python:3.9.4-slim AS build

# set work directory
WORKDIR /app

# set env variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
ARG app_id
RUN echo "app_id = $app_id"
ENV APPLICATION_ID $app_id

# install dependencies
COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

# copy source code
COPY src /app

EXPOSE 8080

CMD python -m uvicorn main:app --reload --host 0.0.0.0 --port 8080
```

requirements.txt

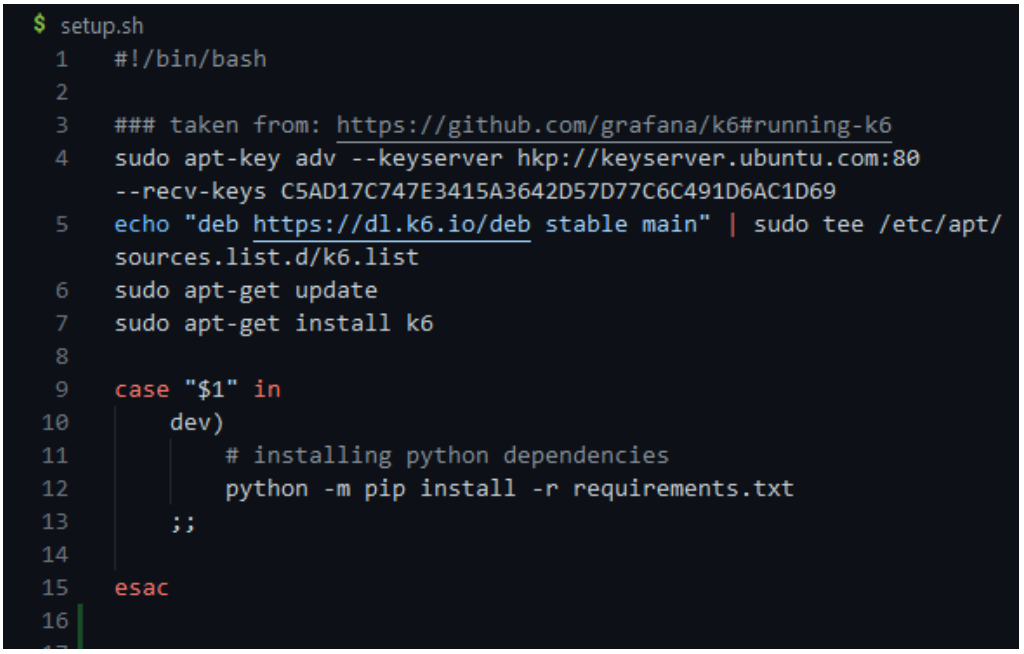
Arquivo onde estão as dependências usadas para executar as APIs.

requirements.txt

```
fastapi==0.81.0
uvicorn==0.18.3
python-dotenv==0.21.0
```

setup.sh

Arquivo com as instalações iniciais do projeto. Pois é necessário instalar algumas dependências para executar o código localmente.



BIBLIOGRAFIA

KOCH, S. Load balancing and auto scaling with open source HAProxy. Disponível em: <https://blog.stefan-koch.name/2021/05/02/load-balancing-auto-scaling-open-source-haproxy>. Acesso em: 3 set. 2022.

No final das contas: o que é o Docker e como ele funciona? Disponível em: <https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona>. Acesso em: 6 set. 2022.

TRUCCO, C. Docker Compose: O que é? Para que serve? O que come? Disponível em: <https://imasters.com.br/banco-de-dados/docker-compose-o-que-e-para-que-serve-o-que-come>. Acesso em: 15 set. 2022.

MONITORA, E. Load balance: o que é, qual a função e como aplicá-lo. Disponível em: <https://www.monitoratec.com.br/blog/load-balance/>. Acesso em: 30 set. 2022.

MWAURA, W. API performance testing with k6. CircleCI, 19 jan. 2022. Disponível em: <https://circleci.com/blog/api-performance-testing-with-k6/>. Acesso em: 10 out. 2022

LOUSADA, F. B. Introdução ao Shell Script no Linux. Disponível em: <https://www.devmedia.com.br/introducao-ao-shell-script-no-linux/25778>. Acesso em: 17 out. 2022.

FICHA DE ATIVIDADE PRÁTICA SUPERVISIONADA

DANIEL LIBERATO - RA T0740B-4

UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Daniel Liberato de Jesus **TURMA:** 08/CC8Q12 **RA:** T0740B-4

CURSO: Ciência da Computação **CAMPUS:** CAMPINAS-SWIFT **SEMESTRE:** 8º **TURNO:** NOTURNO

CÓDIGO DA ATIVIDADE: 7785 **SEMESTRE:** 8º **ANO GRADE:** 2019/1

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09/2022	Reunião em grupo	6	Daniel Liberato		
03/09/2022	Pesquisa sobre HaProxy	5	Daniel Liberato		
06/09/2022	Pesquisa sobre Docker e Docker Compose	6	Daniel Liberato		
12/09/2022	Reunião em grupo	6	Daniel Liberato		
17/09/2022	Pesquisa sobre Auto Scaling	5	Daniel Liberato		
24/09/2022	Configuração do teste de carga com K6	5	Daniel Liberato		
26/09/2022	Reunião em grupo	6	Daniel Liberato		
03/10/2022	Criando ambientes no docker Compose	5	Daniel Liberato		
07/10/2022	Criando configurações do HaProxy	5	Daniel Liberato		
10/10/2022	Reunião em grupo	5	Daniel Liberato		
17/10/2022	Teste de adição de recursos	4	Daniel Liberato		
21/10/2022	Teste de Remoção de recursos	4	Daniel Liberato		
24/10/2022	Teste de resiliência	6	Daniel Liberato		
31/10/2022	Elaboração do relatório	7	Daniel Liberato		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 75

AVALIAÇÃO: _____ Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

_____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Felipe Scherer TURMA: CC8P12 RA: D88HJE-1

CURSO: Ciência da Computação CAMPUS: Campinas - Swift SEMESTRE: 8° TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77B5 SEMESTRE: 8° ANO GRADE: 4°

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09	reunião em grupo	6	Felipe		
03/09	pesquisa sobre HaProxy	5	Felipe		
06/09	pesquisa sobre docker	6	Felipe		
13/09	reunião em grupo	6	Felipe		
17/09	pesquisa sobre auto scaling	5	Felipe		
24/09	pesquisa sobre K6	5	Felipe		
26/09	reunião em grupo	6	Felipe		
03/10	Criando ambiente no docker	5	Felipe		
07/10	Configurando ha proxy	5	Felipe		
10/10	reunião em grupo	5	Felipe		
17/10	teste de ~~edição~~ de recursos	4	Felipe		
21/10	teste de ~~edição~~ remover recursos	4	Felipe		
24/10	teste de resiliência	6	Felipe		
31/10	elaboração do relatório	7	Felipe		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 75

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ___/___/_____

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Jehan M. M. Dias TURMA: CC8P12 RA: N465CC-1

CURSO: Ciência da Computação CAMPUS: Campinas - Swift SEMESTRE: 8º TURNO: Noturno

CÓDIGO DA ATIVIDADE: 7785 SEMESTRE: 8º ANO GRADE: 4º

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09	Reunião em grupo	6	Jehan		
03/09	Pesquisa sobre HAProxy	5	Jehan		
06/09	Pesquisa sobre Docker	6	Jehan		
12/09	Reunião em grupo	6	Jehan		
17/09	Pesquisa sobre auto scaling	5	Jehan		
24/09	Pesquisa sobre K6	5	Jehan		
26/09	Reunião em grupo	6	Jehan		
03/10	Criando ambiente no docker	5	Jehan		
07/10	Configurando HAProxy	5	Jehan		
10/10	Reunião em grupo	5	Jehan		
17/10	Teste de adição de recursos	4	Jehan		
21/10	Teste de remover recursos	4	Jehan		
24/10	Teste de resiliência	6	Jehan		
31/10	Elaboração do relatório	7	Jehan		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 75

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/_____

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

KELLY DENA - RA D912JB-8

UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Kelly Regina Dena da Silva **RA:** D912JB8

TURMA: CC8P12

CAMPUS: Swift **SEMESTRE:** 8º **TURNO:** Noturno

CÓDIGO DA ATIVIDADE: 77B5 **SEMESTRE:** 8º **ANO GRADE:** 2022

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
		3	Kelly	3	
01/09/2022	Reunião em grupo	6	Kelly	6	
02/09/2022	Pesquisa sobre o docker	3	Kelly	3	
04/09/2022	Intalação das ferramentas necessárias	5	Kelly	5	
06/09/2022	Pesquisa sobre HaProxy	6	Kelly	6	
12/09/2022	Reunião em grupo	5	Kelly	5	
17/09/2022	Pesquisa auto scaling	5	Kelly	5	
24/09/2022	Pesquisa sobre K6	6	Kelly	6	
26/09/2022	Runião com o grupo	5	Kelly	5	
03/10/2022	Criação do ambiente do docker	5	Kelly	5	
07/10/2022	Configuração do HaProxy	5	Kelly	5	
10/10/2022	Reunião em grupo	4	Kelly	4	
17/10/2022	Teste de adição de recursos	4	Kelly	4	
21/10/2022	Teste de remover recursos	6	Kelly	6	
24/10/2022	Teste de resiliencia	7	Kelly	7	
31/10/2022	Elaboração do relatório				

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS 75

AVALIAÇÃO: _____ Aprovado ou Reprovado

NOTA: _____

DATA: ___/___/_____

_____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIP
UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Vinicius Guidi Rossi TURMA CC8P12 RA N386BH-8

CURSO: Ciência da Computação CAMPUS: Swift SEMESTRE: 8º TURNO: noturno

CÓDIGO DA ATIVIDADE: 77B5 ANO GRADE: 4º ano

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09	Reunião em grupo	6	Vinicius Rossi	6	
03/09	Pesquisa sobre Haproxy	5	Vinicius Rossi	5	
06/09	Pesquisa sobre Docker	6	Vinicius Rossi	6	
12/09	Reunião em grupo	6	Vinicius Rossi	6	
17/09	Pesquisa sobre auto scaling	5	Vinicius Rossi	5	
24/09	Pesquisa sobre K6	5	Vinicius Rossi	5	
26/09	Reunião em grupo	6	Vinicius Rossi	6	
03/10	Criando ambiente docker	5	Vinicius Rossi	5	
07/10	configurando Haproxy	5	Vinicius Rossi	5	
10/10	Reunião em grupo	5	Vinicius Rossi	5	
17/10	Teste de adição de recursos	4	Vinicius Rossi	4	
21/10	Teste de remover recursos	4	Vinicius Rossi	4	
24/10	Teste de resiliencia	6	Vinicius Rossi	6	
31/10	Elaboração do relatório	7	Vinicius Rossi	7	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 75 H

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/_____

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO