Universidade do Minho

Virtualização de Redes

Trabalho Prático Nº 2

Docker Micro Services

André Vieira A78322

João Freitas A74814

João Mendes A71862

April 30, 2020

Índice

Índice de Imagens

Acrónimos

Docker Docker

HTTP HTTP

FTP FTP

AUTH AUTH

BD BD

PostGreSQL POST

yaml yaml

OAuth OAuth

containers containers

autentication service auth-serv

token token

file server file

Chapter 1

Introdução

Neste relatório iremos expor o processo de desenvolvimento do trabalho prático nº2 da Unidade Curricular de Virutalização de Redes. Este trabalho teve como objetivo aprofundar os conhecimentos relativos ao uso de Docker como ferramenta de criação de containers e a utilização de softwares de autenticação semelhantes ao OAuth. Este serviço caracteriza-se pela possibilidade do acesso a um *file server*, após se proceder à validação de um token. Este é gerado, durante o processo de autenticação de um utilizador no sistema, pelo *autentication service*.

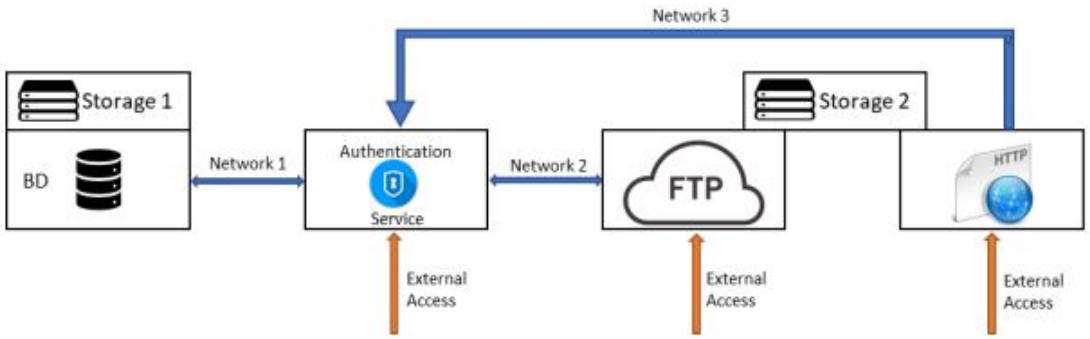

Figure 1.1: Arquitetura do sistema

O processo de funcionamento do sistema é iniciado na autenticação do utilizador com recurso às suas credenciais(email e palavra-passe). Se o utilizador já estiver registado, no sistema, irá ser gerado um token, que após a sua verificação e validação, na base de dados, este irá ter acesso ao *file server*.

A arquitetura do sistema é dividida em duas partes, *frontend* e *backend*. No *frontend* estão inseridos os containers HTTP e FTP. No *backend* temos o serviço de autenticação e a base de dados (construída em PostGreSQL) do sistema, dentro dos containers AUTH e BD, respetivamente.

Como objetivo, é pretendido o desenvolvimento de um sistema capaz de realizar a sua montagem de uma forma automática com recurso á utilização da ferramenta Docker.

Chapter 2

Conceitos Teóricos

2.1 Docker

O conceito de virtualização surge na década de 60, sendo que a sua propagação se deu mais fortemente a partir de 1970, mas só nos anos 2000 começa a ser amplamente utilizada, devido a um maior crescimento do nível computacional. O conceito surgiu devido a ser bastante dispendioso e difícil de modificar um aplicação para correr em diferentes Sistemas Operativos(SOs).
Existem vários tipos de virtualização.

- **Servidores**- Tipo de virtualização mais comum, onde um servidor é dividido em vários ambientes virtuais onde cada um desempenha uma função diferente.
- **Armazenamento**- Chama-se virtualização de armazenamento quando uma única fonte de dados é dispersa por vários locais. A dado de exemplo temos as partições dos discos e o armazenamento distribuído, ou seja,
- **Redes**- Denominado por virtualização de funções de redes(NFV), separa as principais funções de uma rede para as distribuir pelos diferentes ambientes virtuais.
- **Containers**- Irão ser abordados no decorrer deste trabalho

2.1.1 Exemplos de virtualizações

2.1.1.1 Máquinas virtuais

As máquinas virtuais são *computadores dentro de computadores* que são executadas como qualquer outro programa através de ficheiros denominados de **Imagens**. A principal desvantagem está no facto desta solução ocupar muitos recursos do sistema, por outro lado esta solução isola a máquina virtual do sistema **host** sendo, desta forma, ideal para testar outros sistemas operativos, incluindo versões beta, aceder a dados infetados com vírus, criar cópias de segurança de sistemas operativos e executar software ou aplicações em sistemas operativos para os quais não foram originalmente concebidos.

2.1.2 Containers

Ao contrário do exemplo anterior, onde tanto o sistema operativo bem como o computador eram virtualizados, apenas o sistema operativo é virtualizado o que leva a que em termos de recursos seja mais leve e mais rápido a executar.

Cada container em execução partilha o sistema operativo do host com qualquer outro container em execução, mesmo sendo executados como sendo processos isolados de recursos.

Docker é a plataforma mais conhecida que permite a criação de containers, isto porque possibilita o suporte para diferentes sistemas operativos, tais como: Ubuntu, Fedora, CentOS, Windows, etc.

Através de um ficheiro de configuração designado por *dockerfile* é possível criar um container.

```
FROM httpd:latest
RUN apt-get update && apt-get install -y apt-utils
RUN apt-get install -y iputils-ping
RUN apt-get install -y net-tools
RUN apt-get install -y nginx git python-setuptools python-dev
RUN apt-get install -y python3
RUN apt-get install -y python3-pip
RUN pip3 install httpserver
RUN pip3 install flask
RUN pip3 install requests
RUN mkdir /home/code
WORKDIR /home/code
RUN mkdir /home/code/src
RUN mkdir /home/code/Templates
COPY /src/*.* /home/code/src/
COPY /Templates/*.* /home/code/Templates/
COPY my-httpd.conf ./
COPY script.sh ./
RUN chmod +x script.sh
ENTRYPOINT ["/home/code/script.sh"]
```

Figure 2.1: Dockerfile exemplo

Para além disso temos ainda o *docker-compose*, que é um ficheiro em formato yaml que nos permite executar vários containers ao mesmo tempo. É possível também incluir alguns *features* para a configuração de cada um dos containers.

```
auth:
  build: './docker/auths'
  container_name: auth
  volumes:
    - data-authRedes:/var/lib/authRedes/data
  ports:
    - '5000:5000'
  networks:
    - Network1
    - Network2
    - Network3
  restart: always
  depends_on:
    - bd
```

Figure 2.2: Excerto docker-compose

2.2 OAuth

O OAuth é um protocolo ou estrutura de autirização que permite que as aplicações obtenham acesso limitado às contas dos usuários em serviço HTTP. Por acesso limitado, queremos dizer que a

aplicação apenas tem acesso a informação pessoal, como nome, o email ou a fotografia, se for necessário, e nunca a acesso privado, ou seja, a *password* do utilizador para aquele serviço. O OAuth não partilha as credenciais dos utilizadores, pois usa *tokens* para proceder ao pedido de autorização para provar a identidade entre consumidores e provedores de serviços.

O servidor que implementa o protocolo OAuth é capaz de determinar os tipos de *scope* que fornece, ou seja, quais partes da informação do utilizador fornece. Pode-se, por exemplo, definir um *scope* para dar acesso apenas ao *email* do utilizador e outro para dar acesso a toda a informação que temos.

Chapter 3

Solução

O desenho da nossa solução iniciou-se com a decisão das imagens que iriamos utilizar para cada container. Tendo em conta os requisitos do enunciado decidimos nas seguintes imagens.

- BD- postgres:latest
- HTTP- httpd:latest
- FTP- stilliard/pure-ftpd
- AUTH- python:latest

De seguida tivemos que decidir a linguagem que iriamos utilizar para desenvolver cada um dos serviços, sendo que a escolhida foi **Python** visto que esta disponibiliza uma série de bibliotecas que irão ser úteis para a resolução do enunciado.
Por fim passamos ao desenvolvimento da solução em si.

Chapter 4

Implementação

De um modo geral o funcionamento do sistema pode ser descrito por: o utilizador acede à página principal onde se pode registar ou fazer login.
Se quiser registar precisa de fornecer um email, que ainda não tenha sido usado, e uma password.
De seguida o container HTTP irá enviar essa informação para o container AUTH que irá registar o utilizador no container BD, por fim o utilizador é redirecionado para a página **Login**.
Na página **Login** o utilizador irá novamente fornecer um email e uma password que mais uma vez é enviado para o container AUTH que irá, através da BD, verificar se a informação fornecida é válida.
Se esta for válida o container AUTH irá gerar um token que vai ser alocado ao utilizador que se está a tentar autenticar, como forma de substituir o modo de autenticação do protocolo OAuth, como foi dito anteriormente em relação ao funcionamento do mesmo. Para gerar esse token é utilizado o email, a password e o tempo atual, passando estes três parâmetros por uma função de *hashing*, sendo o resultado o token utilizado pelo utilizador em questão.
De acordo com este funcionamento geral optamos pela utilização de uma BD **PostgreSQL**. Para fazermos a ligação entre o AUTH e BD o grupo necessitava de uma biblioteca que fosse capaz de realizar queries **SQL** a partir do AUTH para tal foi encontrada e utilizada a biblioteca *psycopg2*. Para o *autentication service* foram criados servidores *http*. Para a criação desses foi utilizada a framework **Flask**, que permite a criação de endpoints http e de uma aplicação web com relativa facilidade, sendo que as páginas apresentadas foram criadas em *html*.
Cada endpoint do *autentication service* recebe pedidos do tipo **POST**, respondendo com um código correspondente á resposta ao pedido.

- Código 200- OK
- Código 400- Existing User
- Código 404- User not found
- Código 500- Server error

Após a verificação do pedido, é feita a obtenção da resposta a devolver, fazendo, para isso acesso a BD para criar, apagar ou verificar certos dados. No caso do serviço HTTP, é este que é apresentar as páginas ao utilizador. Sendo que, através da biblioteca **requests** do **Python**, irá ser realizado um pedido *http* ao *autentication service*, sendo então analisada a resposta e uma decisão tomada de acordo com esta.

Foi utilizado *flash* para fazer o envio de mensagens de erro para a interface, avisando o utilizador de certas situações, como por exemplo, credenciais erradas no login.

Chapter 5

Resultados e Discussão

São apresentados de seguida as imagens referentes a aplicação web criada:

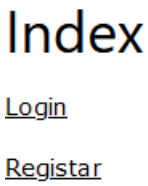

Figure 5.1: Página inicial

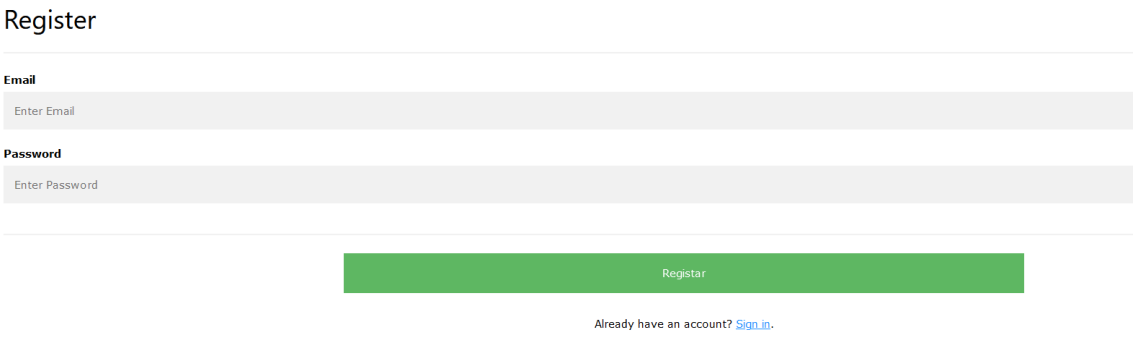

Figure 5.2: Registo

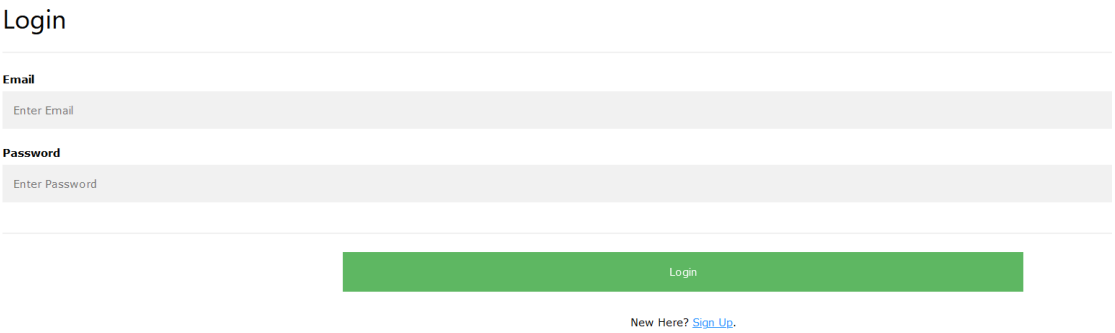

Figure 5.3: Login

Figure 5.4: Main Page

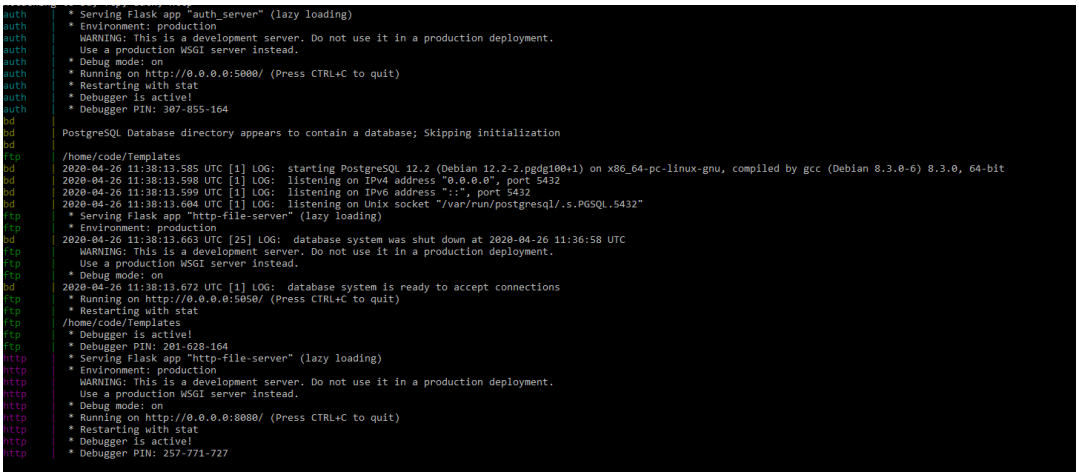

Figure 5.5: Logs da criação das páginas e bd

Como é possível verificar, a parte visual da aplicação não foi muito trabalhada durante o desenvolvimento deste projeto, visto que, não era esse o objetivo do mesmo. É também possível observar na imagem 5.5, que todos os containers criados(HTTP, BD, AUTH, FTP) estão operacionais. Como trabalho futuro o grupo iria criar uma interface visual mais apelativa e interativa para o utilizador.

Chapter 6

Conclusão

A solução implementada pelo grupo é caracterizada por permitir a confidencialidade dos dados do utilizador tanto no momento de registo como no momento do *login*. Sendo que a solução apresenta vária componentes necessárias para que isto seja possível, como por exemplo, as páginas *web* criadas para que o utilizador possa introduzir os seus dados, todo o backend implementado para poder adquirir esses dados que o utilizador fornece e guarde na base de dados e posteriormente gerar um token quando este efetua o *login*.

Infelizmente o resto da solução desenvolvida pelo grupo não se enquadra com a solução prevista pelo docente porque só reparamos nessa divergência no fim da implementação, quando o grupo já não tinha disponibilidade para corrigir, apesar deste revês o grupo está satisfeito com o trabalho realizado.

Em suma, concluímos que o resultado final do nosso projeto é bom, tendo sido conseguido implementar um sistema robusto e persistente com auxilio da ferramenta Docker, que permite apenas ao utilizador ter acesso aos serviços de *http* e autenticação, e estando impossibilitado de aceder à BD. Sendo que caso utilizador tente aceder á página *http*, não lhe será permitido o acesso sem a validação do respetivo token.

Consequentemente conseguimos responder com sucesso à maioria dos requisitos do enunciado, e assim foi possível consolidar os conhecimentos obtidos na unidade curricular, através do desenvolvimento do projeto.