



Universidade do Minho
Escola de Engenharia

Gestão e Virtualização de Redes
ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA
2021/2022

(Docente: Joaquim Melo Henriques Macedo)

08 de fevereiro de 2022

Trabalho Prático 2

Docker & Microservices

Rui Filipe Ribeiro Freitas - pg47639@alunos.uminho.pt

Tiago João Pereira Ferreira - pg47692@alunos.uminho.pt



Índice

Lista de figuras.....	3
Introdução	4
Fase 1	5
Fase 2	5
Fase 3	5
Fase 4	5
Desenvolvimento.....	6
1. Serviço de base de dados	6
2. Serviço de autenticação	7
3. Serviço HTTP	8
Testes e análise de resultados.....	10
Conclusão.....	12

Lista de figuras

Figura 1 – Docker-compose DataBase	6
Figura 2 - bcrypt.....	7
Figura 3 - JWT (JSON Web Token)	7
Figura 4 - Docker-compose Autenticação	8
Figura 5 - Dockerfile Autenticação	8
Figura 6 - Token gerado.....	8
Figura 7 - Ficheiro logs.....	9
Figura 8- Docker-compose Http	9
Figura 9 - Dockerfile Http	9
Figura 11 -Página Registo	10
Figura 12 - Página Login.....	10
Figura 10 - Página User	11
Figura 13 - Página Admin.....	11

Introdução

No âmbito da unidade curricular de Gestão e Virtualização de Redes no módulo de Virtualização de redes foi apresentado pelo docente o trabalho prático número 2 ao qual este relatório diz respeito.

O principal objetivo deste relatório é aprender um pouco mais sobre o que são dockers e como podemos tirar partido destes através da implementação de microserviços. Este relatório está dividido em secções que correspondem às fases apresentadas no enunciado.

Os principais serviços a serem implementados são um serviço de autenticação e 2 servidores de ficheiros. O serviço de autenticação deve ser similar ao Oauth mas numa versão mais simplificada. O serviço deve retornar um token após uma correta autenticação do utilizador. Este token deve então ser usado para aceder aos servidores de ficheiros e realizar o *download* e *upload* destes. O servidor de autenticação deve também guardar os tokens gerados numa base de dados.

De modo a sermos capazes de cumprir com o que é pedido na realização deste trabalho prático foi necessário algum investimento de tempo na aprendizagem sobre tudo o que envolve dockers, assim como alguma leitura da sua documentação pois sem isso não seria possível a realização este trabalho.

Fase 1

A primeira fase deste trabalho tem como objetivos:

- Implementação de um serviço de autenticação que fosse capaz de comunicar com uma base de dados;
- Os serviços de FTP e HTTP devem ser capazes de comunicar com o serviço de autenticação implementado;
- O acesso externo apenas pode ser feito através do serviço de autenticação ou do HTTP;
- A base de dados deve ter persistência;
- O FTP e HTTP devem partilhar o seu sistema de ficheiros.

Fase 2

A segunda fase deste trabalho tem como objetivos:

- Mudar o tipo de autenticação usado pelo HTTP, ou seja, utilizar um tipo de autenticação que redireciona o utilizador para o serviço de autenticação, tendo como base o tipo de autenticação OAuth.

Fase 3

A terceira fase deste trabalho tem como objetivos:

- Criar um reverse proxy;
- Gerar certificados para obter o https;
- Adicionar bind mounts para gerar ficheiros logs.

Fase 4

A quarta fase deste trabalho tem como objetivos:

- Criar diferentes tipos de utilizadores com diferentes permissões.

Desenvolvimento

Em relação a solução desenvolvida esta destacada em 3 pilares essenciais, o serviço de base de dados que contem a informação dos utilizadores, o serviço de autenticação que permite registar e autenticar utilizadores de uma forma segura e eficaz e o servidor HTTP que apresenta diferentes tipos de conteúdos baseados nas permissões de cada utilizador.

No final de desenvolvidos todos os serviços foi criado um ficheiro Docker-compose de modo que a arquitetura desenvolvida possa funcionar em qualquer computador sem que seja necessário configurá-lo.

1. Serviço de base de dados

Relativamente ao desenvolvimento da base de dados, este foi o serviço mais fácil de implementar, bastando apenas transferir a imagem do MYSQL através do Docker Hub. Em relação ao ficheiro Docker-compose tivemos que colocar os seguintes campos que se encontram na figura seguinte para configurar o serviço de base de dados. De salientar o campo networks que constitui a rede que será utilizada para o serviço de autenticação e a base de dados comunicarem entre si, outro aspeto importante é o campo volumes que permite com que a base de dados tenha persistência.

```
services:
  mysql-db:
    image: mysql:latest
    container_name: db_container
    networks:
      - db_network
    ports:
      - "3306:3306"
    expose:
      - '3306'
    restart: always
    environment:
      MYSQL_DATABASE: db_auth
      MYSQL_ROOT_PASSWORD: 12345
      MYSQL_USER: root
      MYSQL_PASSWORD: 12345
    volumes:
      - db_data:/var/lib/mysql
    cap_add:
      - SYS_NICE # Desabilitar o aviso "mbind: Operation not permitted"
```

Figura 1 – Docker-compose DataBase

2. Serviço de autenticação

Para implementar o serviço de autenticação optamos por usar a framework nodeJS , uma vez que é uma plataforma de desenvolvimento muito utilizada, open-source e com vários aspetos positivos.

Primeiramente criamos um sistema de registo de utilizadores onde é pedido um nome, email e palavra-passe. Depois de criado um utilizador a sua informação é guardada na base de dados de forma segura, uma vez que utilizamos a biblioteca bcrypt (figura 2) para que a password não seja guardada em texto, mas sim encriptada.

```
let hashedPassword = await bcrypt.hash(password, 8);  
console.log(hashedPassword);  
if (name !== "admin") {
```

Figura 2 - bcrypt

Na implementação da função de login optamos por usar a biblioteca jwt(JSON Web Token), que permitiu associar um token a cada utilizador quando este realiza um login com sucesso(figura 3). Este token é guardado como forma de um cookie no browser do ulizador, de modo que quando este efetua o login, o mesmo é redirecionado para o serviço de HTTP que verifica o token e toma diferentes ações dependendo to tipo de utilizador.

```
else {  
  const id = results[0].id;  
  const name = results[0].name;  
  const email = results[0].email;  
  const role = results[0].role;  
  if (results[0].role === 'admin') {  
    console.log("admin")  
    token = jwt.sign({ id, name, email, password, role }, process.env.JWT_SECRET_ADMIN, {  
      expiresIn: process.env.JWT_EXPIRES_IN  
    });  
  } else {  
    console.log("user")  
    token = jwt.sign({ id, name, email, password, role }, process.env.JWT_SECRET, {  
      expiresIn: process.env.JWT_EXPIRES_IN  
    });  
  }  
}
```

Figura 3 - JWT (JSON Web Token)

Em relação ao Dockerfile deste serviço este encontra-se representado na figura 4. A primeira linha representa a imagem do node a ser utilizada e a última linha o comando para executar o serviço.

```
FROM node:alpine

WORKDIR /auth

COPY package.json ./

RUN npm install

COPY . .

CMD ["npm", "start"]
```

Figura 5 - Dockerfile

```
auth_service:
  image: tiago19fp/auth_service
  command: npm start
  container_name: auth_container
  restart: always
  networks:
    - db_network
    - auth_network
  ports:
    - "5000:5000"
  expose:
    - "5000"
  depends_on:
    - mysql-db
```

Figura 4 - Docker-compose

Relativamente ao docker-compose deste serviço encontra-se na figura 5. Como se pode observar é utilizada como imagem a imagem que demos upload para o Docker Hub, configuramos também a porta a ser utilizada que neste caso foi a 5000 e as redes a serem usadas, uma rede para comunicar com a base de dados (db_network) e outra para comunicar com o HTTP (auth_network).

3. Serviço HTTP

A implementação do serviço HTTP baseia-se numa aplicação suportada por o nodeJS, assim como no serviço de autenticação, que apresenta diferentes conteúdos consoante o tipo de utilizador e as suas mesmas permissões.

Para conseguir gerir qual página apresentar aos diferentes utilizadores, foi usado como parâmetro de decisão o token gerado (figura 6) pelo serviço de autenticação, que fica guardado como cookie no browser do utilizador, consoante o tipo de token o utilizador é redirecionado para diferentes páginas.

```
(req.cookies.jwt) {
  try {
    const decoded = await promisify(jwt.verify)(req.cookies.jwt, process.env.JWT_SECRET_ADMIN);
    //console.log(decoded);
    var user = {
      id: decoded.id, name: decoded.name, email: decoded.email, password: decoded.password, role: decoded.role };
    console.log(user);
    req.user = user;
    return next();
  } catch (error) {
    console.log(error);
  }
  try {
    const decoded = await promisify(jwt.verify)(req.cookies.jwt, process.env.JWT_SECRET);
    //console.log(decoded.role);
    const user = {
      id: decoded.id, name: decoded.name, email: decoded.email, password: decoded.password, role: decoded.role };
    //console.log(user);
    var today = new Date();
```

Figura 6 - Token gerado

Existem dois tipos de utilizadores disponíveis, o admin e o user. O admin tem acesso a uma página onde pode ver todos os logins dos diferentes utilizadores, já o user tem acesso a uma página onde encontra as suas informações. Sempre que um utilizador

se conecta a este serviço é guardado num ficheiro de log (figura 7) o seu id, nome, email e timestamp do login, para que o admin possa verificar quem tem tido acesso ao serviço implementado.

Relativamente ao Dockerfile deste serviço (figura 9) é praticamente igual ao do serviço anterior, mudando apenas o WORDIR para /http_

Para implementar a persistência foi adicionado ao Docker-compose (figura 8) o campo VOLUMES de modo a que ficasse guardado os logs do utilizadores, em relação ao ao serviço de autenticação este não tem acesso a rede db_network, mas sim apenas a rede auth_network, uma vez que este apenas comunica esse serviço e não com o serviço de base de dados.

```
id,name,email,last_login
1,Tiago,tiago.ferreira.19@hotmail.com,2022-2-7 17:14:55
1,Tiago,tiago.ferreira.19@hotmail.com,2022-2-7 17:15:51
1,Tiago,tiago.ferreira.19@hotmail.com,2022-2-7 17:18:24
1,Tiago,tiago.ferreira.19@hotmail.com,2022-2-7 17:18:32
```

Figura 7 - Ficheiro logs

```
FROM node:alpine

WORKDIR /http

COPY package.json ./

RUN npm install

COPY . .

CMD ["npm", "start"]
```

Figura 9 - Dockerfile Http

```
http_service:
  image: tiago19fp/http_service
  command: npm start
  container_name: http_container
  restart: always
  networks:
    - auth_network
  ports:
    - "5001:5001"
  expose:
    - "5001"
  depends_on:
    - auth_service
  volumes:
    - http_data:/http
```

Figura 8- Docker-compose Http

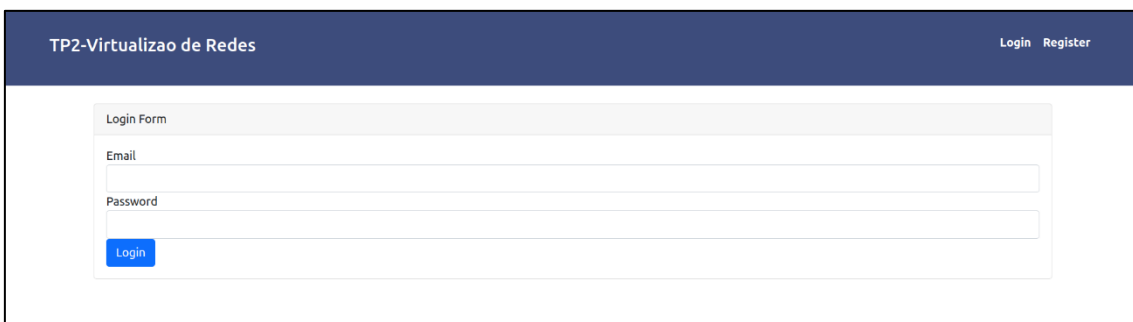
Testes e análise de resultados

Para testar e verificar que a solução desenvolvida pelo grupo estava bem elaborada e que não tinha nenhum problema evidente decidimos adicionar vários utilizadores e um admin de modo a testar todas as funcionalidades desenvolvidas. Nas figuras seguintes encontram-se as diferentes páginas que foram criadas, as duas primeiras imagens correspondem ao login (figura 12) e registo (figura 11) de um utilizador, respetivamente. A figura 10 corresponde a página que um utilizador normal tem acesso, já a figura 13 representa a página que um admin tem acesso.



The screenshot shows a web application interface with a dark blue header. The header contains the text "TP2-Virtualizao de Redes" on the left and "Login Register" on the right. The main content area is white and contains a "Register Form" box. Inside the form, there are four input fields labeled "Name", "Email", "Password", and "Password Confirm". Below these fields is a blue button labeled "Register".

Figura 10 -Página Registo



The screenshot shows a web application interface with a dark blue header. The header contains the text "TP2-Virtualizao de Redes" on the left and "Login Register" on the right. The main content area is white and contains a "Login Form" box. Inside the form, there are two input fields labeled "Email" and "Password". Below these fields is a blue button labeled "Login".

Figura 11 - Página Login

Como se pode verificar pelas figuras acima podemos concluir que obtivemos resultados satisfatórios e sem nenhum erro a destacar.

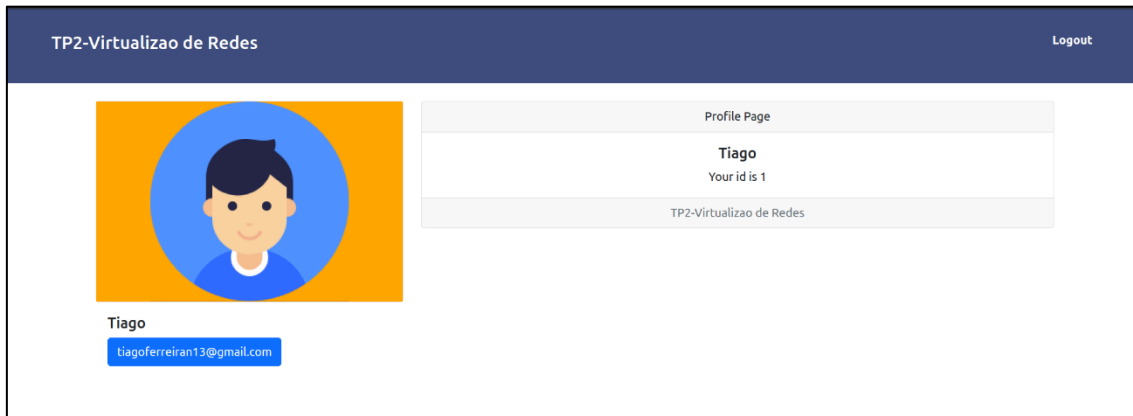


Figura 12 - Página User

TP2-Virtualizao de Redes				Logout
Admin Page				
Last Logins				
Id	Name	Email	Last Login	
1	Tiago	tiago.ferreira.19@hotmail.com	2022-2-7 17:14:55	
1	Tiago	tiago.ferreira.19@hotmail.com	2022-2-7 17:15:51	
1	Tiago	tiago.ferreira.19@hotmail.com	2022-2-7 17:18:24	
1	Tiago	tiago.ferreira.19@hotmail.com	2022-2-7 17:18:32	
6	Bruno	bruno@gmail.com	2022-2-7 17:28:50	
1	Tiago	tiago.ferreira.19@hotmail.com	2022-2-7 17:39:6	
7	Rui	rui@gmail.com	2022-2-7 17:42:44	

Figura 13 - Página Admin

Conclusão

Depois de elaborar o trabalho podemos concluir que foi um processo de aprendizagem importante e bem-sucedido, uma vez que a ferramenta aqui utilizada, o Docker mostrou-se bastante versátil e demonstrou o porquê de cada vez ser mais utilizada no mundo do desenvolvimento de software.

Apesar de não termos desenvolvido todos os objetivos pretendidos, como por exemplo o serviço FTP ou o reverse proxy, podemos afirmar que estamos bastantes contentes com os resultados obtidos uma vez que foi a primeira vez que trabalhamos com o Docker e consequentemente com o docker-compose.