

Virtualização de Redes TP1

Grupo 9

Bruno Chianca Ferreira PG33878
João Rui de Sousa Miguel A74237
Paulo Jorge Machado Guedes A74411

16 Março 2018

1 Introdução

Neste primeiro trabalho prático, foi proposto o desenvolvimento de serviços de autenticação e de e-mail que comunicassem entre si. Para tal foi recomendada a utilização da ferramenta *Docker*, capaz de criar virtualmente estes ambientes, bem como todos os volumes e redes que estes necessitam.

Este trabalho deverá permitir o envio de e-mails por parte dos seus utilizadores. Para tal criou-se uma plataforma capaz de efetuar o registo de novos clientes, plataforma esta que será utilizada para guardar os dados de registo numa base de dados em *PostgreSQL*. Esta permite ainda que se efetue o *login*, possibilitando, finalmente, o envio de mensagens de correio eletrónico.

Além disto, foi desenvolvida a plataforma de e-mail que permite aos utilizadores registados o envio de mensagens. Para tal estes deverão efetuar a autenticação, para que consigam obter um *token* que garanta acesso de envio.

2 Estrutura do projeto

Para criar e inicializar o projeto no ambiente *Docker*, foi feito uso da ferramenta de orquestração *docker-compose*. O projeto consiste numa estrutura de arquivos e diretorias coordenados por um ficheiro do tipo *yml*. Cada diretoria corresponde a um arquivo de construção de imagens *Dockerfile* ou a volumes criados pelo *compose* e adicionados aos *containers* para persistência, portabilidade, e configuração. Abaixo é mostrada a organização dos ficheiros e diretorias. Explicar-se-á de seguida cada uma destas componentes.

Assim, utilizando o *docker-compose.yml* são criados cinco *containers*. Quatro destes containers (*db*, *haproxy*, *mail* e *aut*) usam volumes para manter persistência de dados (no caso do container da base de dados "*db*") ou para efeitos de configuração (*haproxy*, *mail* e *aut*).

O *haproxy* foi implementado como aspeto de valorização, explicado mais à frente no relatório.

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS		NAMES
0c6b409684ff	haproxy:1.7	"/docker-entrypoint..."	6 seconds ago
Up 5 seconds	0.0.0.0:443->443/tcp, 0.0.0.0:8900->80/tcp		haproxy
0f1a03932eb3	namshi/sntp	"/bin/entrypoint.sh ..."	6 seconds ago
Up 5 seconds	25/tcp, 0.0.0.0:32771->1525/tcp		sntp
32afd2447271	vrep1_mail	"docker-php-entrypoi..."	7 seconds ago
Up 5 seconds	0.0.0.0:8890->80/tcp		mail
ecdbf1754a52	vrep1_aut	"docker-php-entrypoi..."	8 seconds ago
Up 6 seconds	0.0.0.0:8889->80/tcp		aut
3caf851596cb	postgres:latest	"docker-entrypoint.s..."	8 seconds ago
Up 7 seconds	0.0.0.0:5432->5432/tcp		db

DRIVER	VOLUME NAME
local	vrep1_autVol
local	vrep1_dbVol
local	vrep1_haproxyVol
local	vrep1_mailVol

Figura 2: Volumes Criados pelo DockerCompose

Figura 1: Containers Criados pelo DockerCompose

Devido à configuração das *networks*, apenas alguns *containers* conseguem comunicar com alguns *containers*, apresentando a seguinte configuração:

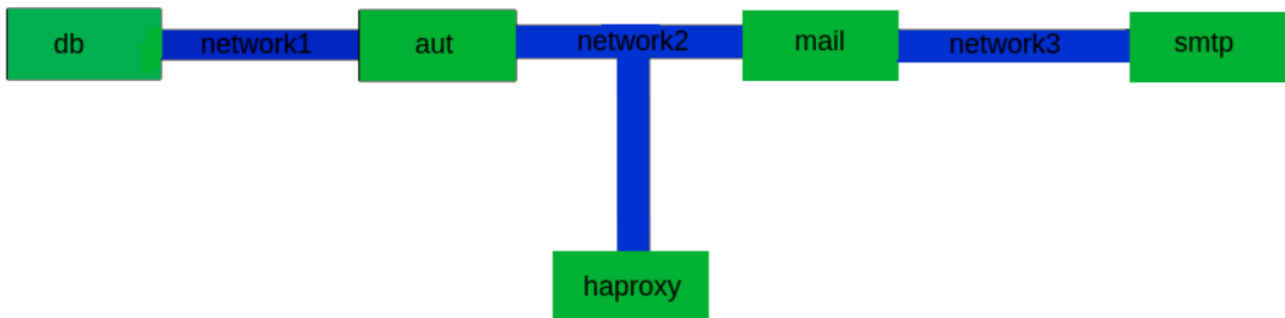


Figura 3: Estrutura das Networks dos Containers Criados

De seguida iremos demonstrar quais *containers* conseguem comunicar com que *containers*.

Container que suporta a Base de Dados (db):

```
root@3caf851596cb:/# ping aut
PING aut (172.51.0.3): 56 data bytes
64 bytes from 172.51.0.3: icmp_seq=0 ttl=64 time=0.130 ms
64 bytes from 172.51.0.3: icmp_seq=1 ttl=64 time=0.140 ms
64 bytes from 172.51.0.3: icmp_seq=2 ttl=64 time=0.137 ms
```

Figura 4: Ping de db para aut

```
root@3caf851596cb:/# ping haproxy
ping: unknown host
```

Figura 5: Ping de db para haproxy

```
root@3caf851596cb:/# ping mail
ping: unknown host
```

Figura 6: Ping de db para mail

```
root@3caf851596cb:/# ping smtp
ping: unknown host
```

Figura 7: Ping de db para smtp

Container que suporta o Serviço de Autenticação (aut):

```
root@ecdbf1754a52:/var/www/html# ping db
PING db (172.51.0.2): 56 data bytes
64 bytes from 172.51.0.2: icmp_seq=0 ttl=64 time=0.161 ms
64 bytes from 172.51.0.2: icmp_seq=1 ttl=64 time=0.124 ms
```

Figura 8: Ping de aut para db

```
root@ecdbf1754a52:/var/www/html# ping haproxy
PING haproxy (172.52.0.4): 56 data bytes
64 bytes from 172.52.0.4: icmp_seq=0 ttl=64 time=0.222 ms
64 bytes from 172.52.0.4: icmp_seq=1 ttl=64 time=0.136 ms
```

Figura 9: Ping de aut para haproxy

```
root@ecdbf1754a52:/var/www/html# ping mail
PING mail (172.52.0.3): 56 data bytes
64 bytes from 172.52.0.3: icmp_seq=0 ttl=64 time=0.200 ms
64 bytes from 172.52.0.3: icmp_seq=1 ttl=64 time=0.136 ms
```

Figura 10: Ping de aut para mail

```
root@ecdbf1754a52:/var/www/html# ping smtp
ping: unknown host
```

Figura 11: Ping de aut para smtp

Container que suporta o Serviço de Proxy (haproxy):

```
root@0c6b409684ff:/# ping db
ping: unknown host
```

Figura 12: Ping de haproxy para db

```
root@0c6b409684ff:/# ping aut
PING aut (172.52.0.2): 56 data bytes
64 bytes from 172.52.0.2: icmp_seq=0 ttl=64 time=0.140 ms
64 bytes from 172.52.0.2: icmp_seq=1 ttl=64 time=0.134 ms
```

Figura 13: Ping de haproxy para aut

```
root@0c6b409684ff:/# ping mail
PING mail (172.52.0.3): 56 data bytes
64 bytes from 172.52.0.3: icmp_seq=0 ttl=64 time=0.186 ms
64 bytes from 172.52.0.3: icmp_seq=1 ttl=64 time=0.134 ms
```

Figura 14: Ping de haproxy para mail

```
root@0c6b409684ff:/# ping smtp
ping: unknown host
```

Figura 15: Ping de haproxy para smtp

Container que suporta o Serviço de E-Mail (mail):

```
root@32afd2447271:/var/www/html# ping db
ping: unknown host
```

Figura 16: Ping de mail para db

```
root@32afd2447271:/var/www/html# ping aut
PING aut (172.52.0.2): 56 data bytes
64 bytes from 172.52.0.2: icmp_seq=0 ttl=64 time=0.195 ms
64 bytes from 172.52.0.2: icmp_seq=1 ttl=64 time=0.130 ms
```

Figura 17: Ping de mail para aut

```
root@32afd2447271:/var/www/html# ping haproxy
PING haproxy (172.52.0.4): 56 data bytes
64 bytes from 172.52.0.4: icmp_seq=0 ttl=64 time=0.133 ms
64 bytes from 172.52.0.4: icmp_seq=1 ttl=64 time=0.137 ms
```

Figura 18: Ping de mail para haproxy

```
root@32afd2447271:/var/www/html# ping smtp
PING smtp (172.53.0.3): 56 data bytes
64 bytes from 172.53.0.3: icmp_seq=0 ttl=64 time=0.209 ms
64 bytes from 172.53.0.3: icmp_seq=1 ttl=64 time=0.134 ms
```

Figura 19: Ping de mail para smtp

Container que suporta o Serviço de SMTP (smtp):

```
root@0f1a03932eb3:/# ping db
ping: unknown host
```

Figura 20: Ping de smtp para db

```
root@0f1a03932eb3:/# ping aut
ping: unknown host
```

Figura 21: Ping de smtp para aut

```
root@0f1a03932eb3:/# ping haproxy
ping: unknown host
```

Figura 22: Ping de smtp para haproxy

```
root@0f1a03932eb3:/# ping mail
PING mail (172.53.0.2): 56 data bytes
64 bytes from 172.53.0.2: icmp_seq=0 ttl=64 time=0.051 ms
64 bytes from 172.53.0.2: icmp_seq=1 ttl=64 time=0.076 ms
```

Figura 23: Ping de smtp para mail

2.1 Base de Dados

Para armazenar os dados dos clientes, optou-se por guardar estas mesmas informações numa base de dados em *PostgreSQL*. Para tal, utilizou-se a imagem *postgres:latest*, do *Dockerhub*, buscando a versão mais recente do *PostgreSQL*. O serviço de base de dados corre sobre um *IP* estático, sendo que o seu acesso pode ser efetuado pela porta 5432.

Decidiu-se criar uma nova base de dados, com o nome de **vr**, para albergar as informações dos utilizadores. Para tal, criou-se também um novo utilizador, **vr**, com a password **vr**, sendo este o administrador da base de dados, que será utilizado aquando a inserção de dados, ou criação das tabelas.

Foi criado um *bash script* "postScript.sh", de modo a criar a base de dados, assim como o utilizador com as permissões necessárias para a modificação desta. Assim, após a criação do *container* da base de dados, apenas é necessário executar os seguintes comandos:

1. `docker cp postScript.sh db:/` (Copia o *script* para o *container* da base de dados);
2. `docker exec -it db /bin/bash` (Abre a *bash* dentro do *container* da base de dados);
3. `./postScript.sh` (Executa o script, criando a base de dados e o utilizador).

As tabelas da base de dados são criadas, quando um utilizador acede à página de autenticação (caso as tabelas ainda não tenham sido criadas), no entanto, a base de dados e o utilizador **vr** necessitam de ser previamente criados.

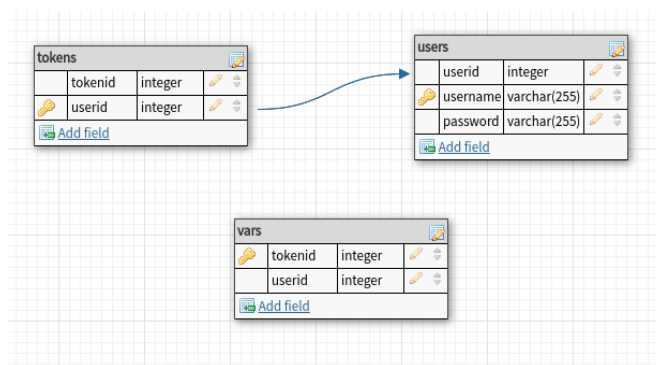


Figura 24: Modelo lógico da Base de dados

A tabela "vars" guarda os valores das variáveis a serem usadas nas outras tabelas, mais concretamente, guarda o ID de cada utilizador, sendo este incrementado sempre que é registado um novo utilizador na tabela "users", e guarda o ID de cada token, sendo incrementado também na tabela "vars" sempre que é feito um novo *login*, criando um par (ID do token, ID do user que fez login). Quando é feito o registo de um novo utilizador, são guardados também os seus dados de *username* e *password*, que serão verificados aquando a tentativa de login. Caso os valores de *username* e *password* estejam corretos no *login*, é criada uma entrada na tabela "tokens" com o ID do username proveniente da tabela "users", e o ID do token. Caso já exista uma entrada com o mesmo utilizador, será atualizado o valor do *tokenid*.

Se por alguma razão o *container* que alberga a base de dados PostgreSQL reiniciar ou desligar, é necessária uma garantia que os dados não sejam perdidos, e que possam ser importados para outro *container*, carregando os dados já existentes noutra base de dados. Para garantir esta persistência e portabilidade, foi utilizado um volume criado no docker-compose, com o nome de dbVol, que importa os dados de uma diretoria da máquina *host*, para a diretoria padrão da base de dados PostgreSQL, dentro do *container*.

2.2 Serviço de Autenticação

Foi decidido realizar o serviço de autenticação utilizando a linguagem PHP. Sendo assim, foi utilizada a imagem **php:7.0-apache** do *Dockerhub* que inicializa o serviço apache aquando a criação do container, mas para a conexão do PHP com a base de dados era necessária instalação do pacote `pdo_pgsql`, assim como as suas dependências, para além da imagem do PHP, de modo a realizar as *queries* à base de dados. Por isso, foi necessária a criação de um *Dockerfile*, para a criação de imagem personalizada baseada na imagem previamente mencionada, com a adição dos novos pacotes.

O serviço de autenticação vai ser responsável por garantir aos utilizadores a existência de uma plataforma simplificada para efetuar registo ou iniciar sessão.

Decidiu-se dividir este serviço em quatro partes:

- `index.php`;
- `sign.php`;
- `login.php`;
- `checkToken.php`.

Para tal, este comunica com a base de dados, sendo que utiliza o método *POST* para enviar as informações sobre os dados do utilizador, juntamente com código desenvolvido em *PHP* para estabelecer este elo de ligação entre os dados introduzidos pelo utilizador e as corretas introduções, validações e autenticações por parte da base de dados.

Entenda-se, assim, que este serviço compreende um servidor apache que redireciona o utilizador através de uma interface gráfica de menu, permitindo a introdução de dados desejados para registo ou início de sessão.

É de salientar que, todos os ficheiros necessários para o funcionamento do serviço de autenticação, são importados da máquina *host* para o *container* através de um volume criado no *docker-compose*, com o nome de `autVol`.

2.2.1 Index.php

Esta será a página acedida por defeito na porta 80 do *container* "aut" (nome do *container* que corre o serviço de autenticação). É apresentada uma interface muito básica, escrita em *HTML* e *CSS*, onde poderá inserir o *username* e *password* do utilizador.

Será disponibilizado duas opções, "*Register*" e "*Login*".

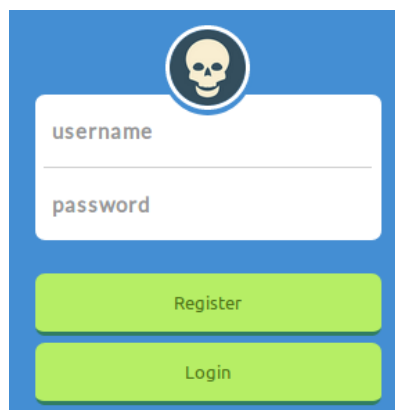
The image shows a web form for user authentication. It has a blue background. At the top center is a circular icon containing a white skull. Below the icon is a white rectangular area containing two input fields. The first field is labeled 'username' and the second is labeled 'password'. Below these fields are two green buttons. The top button is labeled 'Register' and the bottom button is labeled 'Login'.

Figura 25: Página de login

2.2.2 Sign.php

A opção *Register* redirecionará o utilizador para a página *sign.php*, enviando as informações do utilizador (*username* e *password*) para essa mesma página. Será realizada uma conexão à base de dados, verificando se existe algum utilizador com o mesmo *username*, registando caso não exista, reportando o sucesso do registo, ou reportando o insucesso do registo.

De seguida será apresentado o processo de registo de um novo utilizador, assim como as modificações da base de dados, seguido de uma tentativa de registo de um novo utilizador com o mesmo *username*.

Registo válido:

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
(0 rows)

tokenid | userid
-----+-----
0 | 0
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 26: Base de Dados Antes do Registo Válido

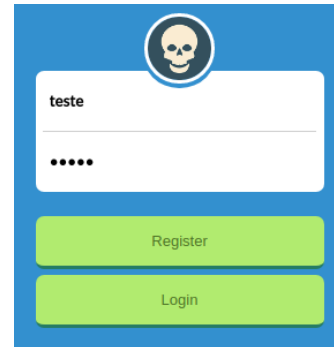


Figura 27: Teste de Registo de Utilizador

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
0 | teste | teste
(1 row)

tokenid | userid
-----+-----
0 | 1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 28: Base de Dados Após Registo Válido

Registado com sucesso

Figura 29: Output do Registo Válido

É de salientar que foi criada uma entrada na primeira tabela (tabela de *users*), com o *userid* da segunda tabela (tabela de *vars*), *username* e *password* introduzidos pelo utilizador, após o qual foi incrementada a variável de *userid* na mesma tabela de *vars*.

Registo Inválido:

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
0 | teste | teste
(1 row)

tokenid | userid
-----+-----
0 | 1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 30: Base de Dados Antes do Registo Inválido

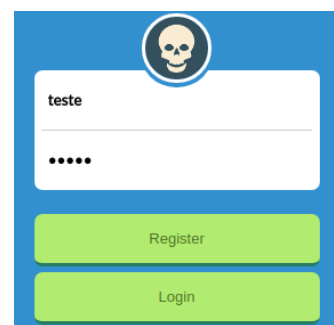


Figura 31: Teste de Registo Inválido de Utilizador


```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      0 |      1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 32: Base de Dados Após Registo Inválido

2.2.3 Login.php

Caso queira fazer *login*, de modo a utilizar o serviço de E-mail, será redirecionado para a página "login.php", que realizará uma conexão à base de dados, de modo a verificar se os dados do utilizador estão corretos, que por sua vez, redirecionará para o serviço de E-mail caso os dados estejam corretos, ou relatando o porquê do seu insucesso.

De seguida será apresentado o processo de *login*, assim como as modificações na base de dados, em 3 casos: caso de insucesso por utilizador inexistente, insucesso por password incorreta e caso de sucesso.

Insucesso por utilizador inexistente:

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      0 |      1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 34: Base de Dados Antes do Login Inválido

Figura 35: Teste de Login Inválido devido a Username

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      0 |      1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 36: Base de Dados Após o Login Inválido

Insucesso por *Password* incorreta:

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      0 |      1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 38: Base de Dados Antes do Login Inválido

Figura 39: Teste de Login Inválido devido a Password

Username de utilizador já existente

Figura 33: Output do Registo Inválido

Não existe utilizador

Figura 37: Output do Login Inválido

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      0 |      1
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 40: Base de Dados Após o Login Inválido

Login efetuado com sucesso:

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
(0 rows)

tokenid | userid
-----+-----
      0 |      0
(1 row)

tokenid | userid
-----+-----
(0 rows)
```

Figura 42: Base de Dados Antes do Login Válido

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
      1 |      1
(1 row)

tokenid | userid
-----+-----
      0 |      0
(1 row)
```

Figura 44: Base de Dados Após o Login Válido

Foi criada uma entrada na última tabela (tabela de "tokens"), com o "userid" do utilizador do qual realizou o login, e com o "tokenid" da segunda tabela (tabela de "vars"), após o qual foi incrementada a variável "tokenid" da mesma tabela "vars". Após o sucesso do login, é redirecionado para um novo *URL*, redirecionando para o servidor de E-mail, com o *token* do utilizador que realizou o *login*. É de realçar que o valor do *token* é enviado através do método *POST*, sendo guardado como variável no envio do e-mail por parte do serviço de e-mail.

2.2.4 CheckToken.php

Como o único *container* que consegue comunicar com a base de dados, é o *container* com o serviço de autenticação, foi criada a página "checkToken.php", que realiza uma conexão à base de dados, verificando após uma tentativa de envio de E-mail, se o *token* corresponde a um utilizador que tenha *login* efetuado. Caso o token seja válido, é redirecionado para a página "mail.php" do serviço de E-mail, explicado de seguida. Caso o token seja inválido, é apresentada uma mensagem de erro. Testes que afetam este serviço serão demonstrados de seguida, juntamente com os testes do envio de E-mail.

Não existe utilizador

Figura 41: Output do Login Inválido

Figura 43: Teste de Login Válido

/mail?token=0

Figura 45: URL após Login Válido

2.3 Serviço de E-mail

O serviço de e-mail consiste num *container* semelhante ao do serviço de autenticação com Apache e PHP.

Foi dividido o serviço em 2 partes:

- index.php
- mail.php

Assim como realizado no serviço de autenticação, todos os ficheiros necessários para o funcionamento do serviço de e-mail, são importados da máquina *host* para o *container* através de um volume criado no docker-compose, com o nome de mailVol.

2.3.1 Index.php

Esta será a página acedida aquando efetuado o *login* no serviço de autenticação. É apresentada uma *interface* muito básica, semelhante à página "index.php" do serviço de autenticação. Nesta página poderá colocar o e-mail do destinatário, o assunto do e-mail, e o texto do mesmo.

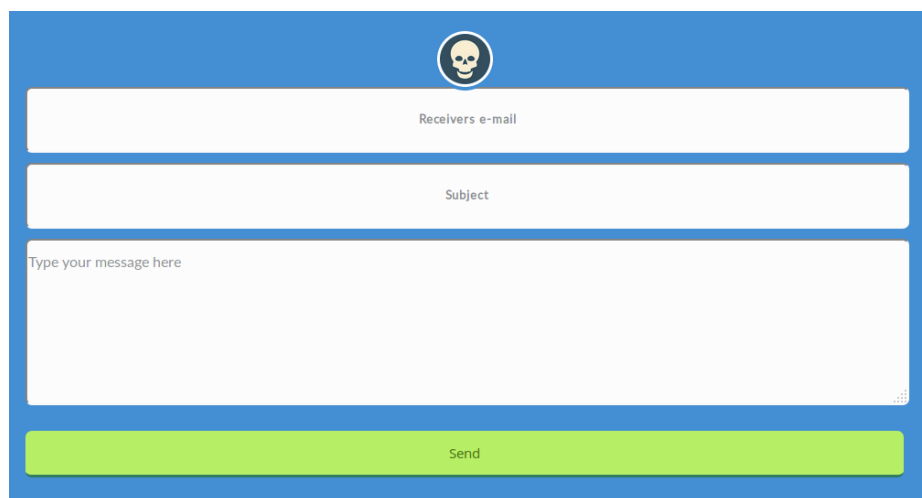


Figura 46: Página de envio de e-mails

Após a tentativa de envio do e-mail, será redirecionado para a página "checkToken.php", do *container* do serviço de autenticação, explicado anteriormente.

2.3.2 Mail.php

Página apenas acedida caso o *token* seja válido, enviando o email para o destinatário colocado no formulário da página "index.php", com as informações da mensagem também colocadas. É apresentado os *logs* do envio do e-mail e uma mensagem a informar o sucesso ou insucesso do processo.

De seguida, será apresentado o processo de verificação do *token* e envio do e-mail, apresentando os valores do *token* enviados para o "checkToken.php", inspecionando a página "index.php".

Token Válido:

Figura 47: Página do serviço E-Mail

```
<form action="/aut/checkToken.php?token=2" method="POST">...</form>
```

Figura 48: Valor do Token

```
vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
0 | teste | teste
(1 row)

tokenid | userid
-----+-----
3 | 1
(1 row)

tokenid | userid
-----+-----
2 | 0
(1 row)
```

Figura 49: Base de Dados No Envio de E-Mail

```
Subject: Teste
Sent to: dnask2412@gmail.com
Message: Isto é um Teste

LOG:
2018-03-14 21:48:45 SERVER -> CLIENT: 220 d30472a9fa40 ESMTP Exim 4.84_2 Wed, 14 Mar 2018
21:48:45 +0000
2018-03-14 21:48:45 CLIENT -> SERVER: EHLO localhost
2018-03-14 21:48:45 SERVER -> CLIENT: 250-d30472a9fa40 Hello mailvrep1_network3
[172.53.0.2]250-SIZE 52428800250-8BITMIME250-PIPELINING250 HELP
2018-03-14 21:48:45 CLIENT -> SERVER: MAIL FROM:<mail@vr-g9.gcom.di.uminho.pt>
2018-03-14 21:48:45 SERVER -> CLIENT: 250 OK
2018-03-14 21:48:45 CLIENT -> SERVER: RCPT TO:<dnask2412@gmail.com>
2018-03-14 21:48:45 SERVER -> CLIENT: 250 Accepted
2018-03-14 21:48:45 CLIENT -> SERVER: DATA
2018-03-14 21:48:45 SERVER -> CLIENT: 354 Enter message, ending with "" on a line by itself
2018-03-14 21:48:45 CLIENT -> SERVER: Date: Wed, 14 Mar 2018 21:48:45 +0000
2018-03-14 21:48:45 CLIENT -> SERVER: To: dnask2412 <dnask2412@gmail.com>
2018-03-14 21:48:45 CLIENT -> SERVER: From: Virt-Redes <mail@vr-g9.gcom.di.uminho.pt>
2018-03-14 21:48:45 CLIENT -> SERVER: Reply-To: Admin <jrsmiguel@outlook.pt>
2018-03-14 21:48:45 CLIENT -> SERVER: Subject: Teste
2018-03-14 21:48:45 CLIENT -> SERVER: Message-ID:
<k6Pev4HNH4Ucon9NBaHcnbvrQfsieeoGaxqE108R8@localhost>
2018-03-14 21:48:45 CLIENT -> SERVER: X-Mailer: PHPMailer 6.0.3
(https://github.com/PHPMailer/PHPMailer)
2018-03-14 21:48:45 CLIENT -> SERVER: MIME-Version: 1.0
2018-03-14 21:48:45 CLIENT -> SERVER: Content-Type: text/plain; charset=iso-8859-1
2018-03-14 21:48:45 CLIENT -> SERVER: Content-Transfer-Encoding: 8bit
2018-03-14 21:48:45 CLIENT -> SERVER:
2018-03-14 21:48:45 CLIENT -> SERVER: Isto é um Teste
2018-03-14 21:48:45 CLIENT -> SERVER:
2018-03-14 21:48:45 SERVER -> CLIENT: 250 OK id=1ewEGL-000056-9z
2018-03-14 21:48:45 CLIENT -> SERVER: QUIT
2018-03-14 21:48:45 SERVER -> CLIENT: 221 d30472a9fa40 closing connection

Message sent!
```

Figura 50: Sucesso No Envio do Email

Note-se que o valor do *token* no *form* é igual ao valor do "tokenid" na última tabela da base de dados, que se refere o utilizador que fez *login* anteriormente.

Token Inválido:

Figura 51: Página do serviço E-Mail

```
<form action="/aut/checkToken.php?token=0" method="POST">...</form>
```

Figura 52: Valor do Token

```

vr=# select * from users; select * from vars; select * from tokens;
userid | username | password
-----+-----+-----
      0 | teste   | teste
(1 row)

tokenid | userid
-----+-----
       3 |      1
(1 row)

tokenid | userid
-----+-----
       2 |      0
(1 row)

```

Figura 53: Base de Dados No Envio de E-Mail

Note-se que o valor do *token* no *form* desta vez é diferente ao valor do "tokenid" na última tabela da base de dados, resultando no insucesso do envio de e-mail.

Token inválido

Figura 54: Insucesso no envio do email

2.4 Servidor SMTP

Para envio de e-mails, foi usado um *container* com servidor SMTP. A imagem *Docker* escolhida foi a imagem **namshi/smtp** que disponibiliza um servidor de *sendmail* na porta 25. Nenhuma configuração adicional foi feita para a demonstração deste projeto.

A demonstração do funcionamento deste servidor, foi demonstrado no teste do serviço anterior, que faz uso do servidor SMTP para enviar E-mails.

2.5 Serviço de Haproxy

De forma a evitar que o usuário final aceda aos serviços através dos ips das máquinas, foi incluído também um serviço de proxy, utilizando o *software* Haproxy. Desta forma, fazendo uso somente de um endereço de *frontend*, o usuário pode escolher entre aceder ao serviço de autenticação ou e-mail fazendo uso do path `/aut` ou `/mail`, respetivamente, e o Haproxy faz o devido reencaminhamento para os serviços correspondentes, escondendo do usuário as informações dos containers. Para o *Haproxy*, foi usada a imagem **haproxy:1.7**

É necessária uma configuração inicial do Haproxy, criando as respetivas regras de reencaminhamento, reencaminhando para o serviço "aut", caso o *url* tenha a expressão regular `/aut`, e reencaminhando para o serviço "mail", caso o *url* tenha a expressão regular `/mail`. Estas configurações são importadas para o *container* através do volume criado no docker-compose "haproxyVol"

3 Desenvolvimento

Explicados todos os aspetos fundamentais do projeto, iremos proceder à explicação do desenvolvimento por parte do grupo de trabalho.

3.1 Projeto Base

Inicialmente foi criado o `docker-compose.yml`, com apenas um *container*, `mail`, com o serviço de base de dados. Procedemos ao desenvolvimento do *bash script* para a criação da base de dados e do utilizador que irá administrar a base de dados, criando assim o ficheiro `postScript.sh`. Foram esboçadas as tabelas com as informações necessárias para o correto funcionamento da aplicação.

Em conjunto com a base de dados, foram adicionadas três redes ao `docker-compose.yml`, `"network1"`, `"network2"` e `"network3"`, que seriam ligadas aos *containers*, permitindo a comunicação entre apenas *containers* específicos. Foi ligado o *container* da base de dados à `"network1"`.

De seguida foi adicionado o *container* `aut`, com o serviço de autenticação, criando-o apenas com a imagem `php:7.0-apache`. Para haver comunicação com a base de dados, foi ligado o novo *container* à `"network1"`. Numa primeira abordagem, foi realizada uma tentativa de comunicação com a base de dados através do pacote `php-pgsql`, não tendo sucesso. Decidiu-se tentar realizar a comunicação através do pacote `pdo-psql`, sendo bem sucedido. Para o sucesso da comunicação com a base de dados, foram atribuídos *IPs* estáticos às redes e aos *containers* nessas redes, sabendo o ip de acesso à base de dados. Foi também decidido que as tabelas da base de dados seriam criadas aquando o acesso à página `"index.php"` do serviço de autenticação, caso ainda não tivessem sido criadas.

Em paralelo com a criação do serviço de autenticação, foi desenvolvido o serviço de E-mail, com apenas o intuito de enviar os respetivos E-mails, sem qualquer tipo de verificação de *token*. Para tal, foram adicionados dois *containers*, `mail` e `smtp`, que albergavam o serviço de E-mail e SMTP, respetivamente. Foram adicionados os *containers* `mail` e `aut` à rede `"network2"`, e os *containers* `mail` e `smtp` à rede `"network3"`. Foram também atribuídos *IPs* estáticos às redes e aos *containers*.

Após o sucesso do envio de um E-mail, foi criada a interface do serviço de E-mail, descrita acima. Tendo em conta que apenas será possível a utilização do serviço se tiver feito login, foi necessária a verificação do *token* enviado pelo serviço de autenticação, mas como o serviço de E-mail não consegue comunicar com a base de dados, foi necessária a criação da página `"checkToken.php"` no serviço de autenticação, para a verificação do *token*.

3.2 Valorização

Tendo concluído o projeto base, decidimos realizar as tarefas de valorização.

Sendo assim, modificamos o método de criação dos volumes, atribuindo variáveis de ambiente às diretorias da máquina *host* de onde a informação dos volumes seria obtida. Com esta modificação, teremos de especificar o caminho absoluto dos volumes. Inicialmente foi criado um ficheiro `".env"` que continham os caminhos padrão dessas mesmas variáveis ambiente, mas estas não reconheciam a variável global `"$PWD"`, que especifica o caminho absoluto até ao ficheiro em questão, que possibilitava um método universal de declarar o caminho até às diretorias padrão, em qualquer *host*. Como alternativa, criamos um ficheiro *bash script* `"export.sh"`, que modifica as variáveis ambiente temporariamente, sendo que a *bash script* já aceita a variável `"$PWD"`. Sendo assim, para executar ficheiro, apenas terá de executar `". export.sh"`, antes de realizar o `"docker-compose up"`. Caso queira declarar outro caminho para os volumes, apenas necessitará de executar o comando `"export {vol_Name}=/path/to/directory"`, sendo que os nomes dos volumes são `"DB"`, `"AUT"`, `"MAIL"`, `"HAPROXY"`, para o volume da base de dados, do serviço de autenticação, do serviço de E-mail e para o serviço de proxy, respetivamente.

De seguida, foi introduzido um novo *container* `"haproxy"` que alberga o serviço de proxy, explicado acima. Foi adicionado este *container* à rede `"network2"`, de modo a comunicar apenas com o serviço de E-mail e autenticação. Após sugestão do professor, foi nos proposto a utilização dos nomes dos *containers*, para o acesso aos respetivos. Sendo assim, foram retirados os *IPs* estáticos das redes criadas, realizando as várias conexões entre os vários *containers* através dos seus nomes.

De forma a adquirir mais conhecimentos de como seria colocar em produção uma aplicação web que faz uso de orquestração de *containers Docker*, foi feita a instalação da aplicação num servidor virtual sob um registo de domínio. Como um integrante do grupo já possuía o domínio *haab.space*, foi necessário somente criar mais um *A record* para um subdomínio.

Para o servidor onde o *Docker* foi instalado, foi usado um servidor alugado da *Vultr.com* com uma máquina virtual *Ubuntu Server 16.04*. Para testes, a ferramenta está então disponível por alguns dias em <http://vrmail.haab.space/aut>.

Até agora, foi sempre acedido aos serviços de forma insegura (*http*). Como o último aspeto da valorização, foi habilitado o uso de *https*, usando o serviço de distribuição de certificados *Let's Encrypt*¹, e o certificado foi adicionado ao *Haproxy* seguindo os seguintes passos:

1. Instalar o Certbot da EFF² no host do Docker.

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get update
$ sudo apt-get install certbot
```

2. Usar o Certbot para baixar e criar um certificado e seguir os passos conforme instruções

```
$ sudo certbot certonly -d vrmail.haab.space
```

3. Juntar o certificado e a chave em um arquivo só

```
$ sudo cat /etc/letsencrypt/live/vrmail.haab.space/fullchain.pem \\
/etc/letsencrypt/live/vrmail.haab.space/privkey.pem > \\
/etc/haproxy/certs/letsvr.pem'
```

4. Copiar o certificado para a diretoria de configuração do haproxy e adicionar a configuração do certificado.

```
bind *:443 ssl crt /usr/local/etc/haproxy/letsvr.pem
```

A figura seguinte demonstra as informações básicas referentes ao certificado.

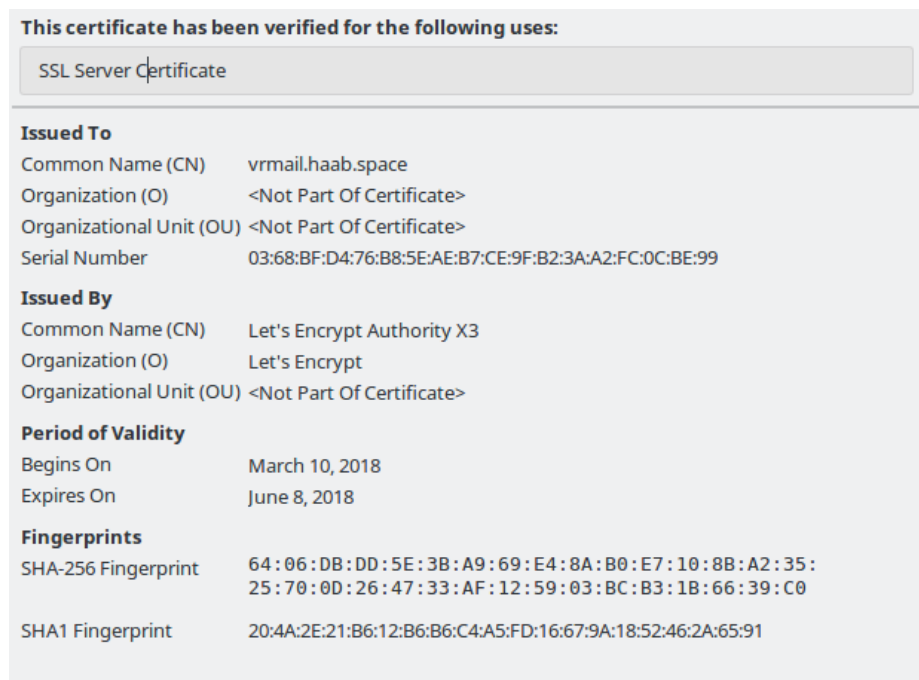


Figura 55: Informações sobre o certificado

Concluídos estes passos, é possível aceder ao serviço de forma segura, através de *https* (<https://vrmail.haab.space/aut>).

¹<https://letsencrypt.org/>

²<https://certbot.eff.org>

4 Conclusão

Este trabalho ajudou a entender tanto o método de funcionamento de ambientes virtualizados sob a plataforma *Docker*, mais em específico as comunicações entre *containers*, como o método de funcionamento de *tokens* utilizado na comunicação entre vários serviços no dia-a-dia.