Universidade do Minho

Virtualização de Redes

Trabalho Prático Nº 1

Familiarização com Docker

Nuno Leite
A70132

Bruno Carvalho
A76987

Gonçalo Duarte
A77508

6 de Abril de 2019

Conteúdo

Lista de Figuras

1 Introdução

A resolução deste trabalho foi proposta no âmbito da Unidade Curricular de Virtualização de Redes, inserida no perfil de especialização de Gestão e Virtualização de Redes, lecionada no segundo semestre do quarto ano do Mestrado Integrado de Engenharia Informática.

O presente trabalho consiste na implementação de um serviço de autenticação e um serviço de email. O serviço de autenticação deverá funcionar de forma semelhante ao *Oauth*. De forma a complementar o trabalho é pedido que sejam criados *containers docker*, que deverão comunicar entre si. Os serviços, armazenamento e redes deverão ser configurados de acordo com a seguinte figura.

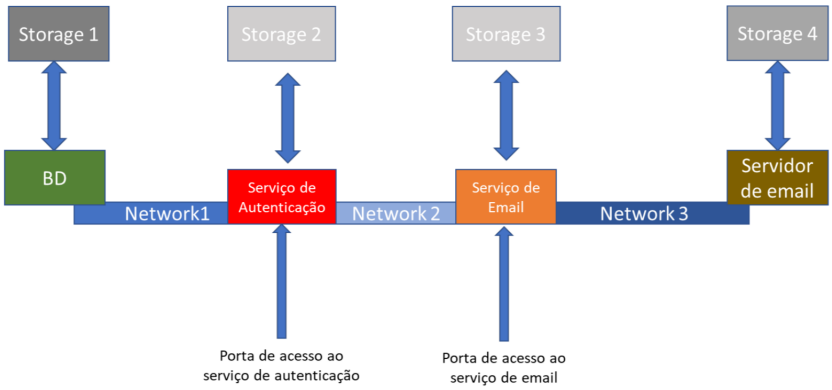

Figura 1: Configuração pretendida

De seguida, deverão ser criadas imagens *docker*,construídas através do *dockerfile*, contendo as aplicações requeridas inicialmente (serviço de email e serviço de autorização). Por fim, o objetivo será produzir um ficheiro *docker-compose.yml* que poderá ser executado em qualquer máquina, de forma a instalar automaticamente toda a arquitetura desenhada.

De modo a satisfazer todos os requisitos especificados e os objetivos mencionados, o relatório apresenta a seguinte estrutura de exposição do trabalho realizado:

- Conceitos Teóricos, onde serão, resumidamente, abordados certos conceitos necessários à melhor compreensão do trabalho desenvolvido.
- Solução, onde será explicada a solução para a elaboração da ferramenta proposta.
- Implementação, onde serão explicitadas todas as fases de desenvolvimento da ferramenta.
- Manual de utilização, que contém as diretrizes para facilitar a utilização da ferramenta.
- Resultados e Discussão, onde constam os resultados obtidos pelo grupo bem como a sua apreciação
- Conclusão

2 Conceitos teóricos

Como já foi mencionado anteriormente, esta secção tem o intuito de explicar certos conceitos que são relevantes para a melhor perceção do trabalho realizado. No contexto deste trabalho, o grupo decidiu oportuno abordar os conceitos que se seguem.

2.1 Docker

O Docker é um programa que desempenha uma virtualização a nível do sistema operativo. O Docker é usado para executar pacotes de software designados *containers*. Os *containers* são isolados uns dos outros em que cada um contém sua própria aplicação, ferramentas, librarias e ficheiros de configuração.

Um *container* é uma unidade standard de software que integra código e todas as suas dependências, para que uma aplicação funcione rapidamente e confiavelmente de um ambiente de computação para outro. Um Docker *container* é um pacote de software leve, independente e executável que contém tudo o que é necessário para correr uma aplicação.

Todos os *containers* são executados pelo mesmo kernel do sistema operativo e por isso são mais leves do que as máquinas virtuais. Os *containers* são criados a partir de "imagens"que especificam, precisamente, o seu conteúdo.

Graças a este sistema de *containers*, os programadores estão seguros de que a aplicação irá ser executada em qualquer outra máquina, independentemente das configurações personalizadas que essa máquina pode ter e que a difiram da máquina que foi usada, originalmente, para escrever e testar o código.

2.1.1 Networks

Uma das razões dos *docker containers* serem tão poderosos prende-se ao facto de estes poderem ser conectados entre si. Existem diversos *drivers* de rede e cada um deles serve um propósito diferente. *Bridge*, *host*, *overlay*, *macvlan* e *none*, são os driver disponíveis, excluindo aqueles que podem ser obtidos por terceiros.

Durante a realização deste trabalho será sempre utilizado o *driver Bridge* uma vez que este é usado habitualmente quando as aplicações que correm em *containers* isolados precisam de comunicar entre si.

2.1.2 Dockerfile

O *docker* pode construir imagens automaticamente lendo as instruções de um *dockerfile*. O *dockerfile* é um documento textual que contêm todos os comandos que um utilizador necessita para montar uma imagem.

Para construir automaticamente a imagem basta introduzir o comando abaixo indicado, sendo que *PATH* é o caminho para a pasta onde se encontra o dockerfile pretendido.

```
docker build [PATH]
```

Figura 2: Comando dockerfile

3 Solução

Com o intuito de esclarecer a secção de implementação passamos agora a explicar a solução equacionada pelo grupo.

Inicialmente começamos por desenvolver o serviço de email e o serviço de autorização utilizando *Node.js*, ambas as aplicações desenvolvidas irão fazer redirecionamento entre si, uma vez que quando um novo utilizador deseja utilizar o serviço de email necessita de ser redirecionado para o serviço de autorização de maneira a obter o *token* que permitirá o envio de email. Esta forma de autorização é inspirada no padrão *OAuth* que permite aos utilizadores fazer *logon* em sites de terceiros através das suas contas de *google*, *facebook*, etc.

Após a conclusão do desenvolvimento destes dois serviços, o grupo construiu o *dockerfile* que iria fazer a build automática destes dois serviços, transformando assim estes dois em imagens *docker*, baseando-se na imagem *node:10* disponibilizada pelo *docker hub*. Ambas as imagens construídas irão mais tarde ser utilizadas no *docker compose*.

Finalmente como servidor de email o grupo decidiu utilizar a imagem *namshi/smtp* e para o volume responsável pela persistência de informação a imagem *mongo:3.0*. Desta forma encontramo-nos agora capazes de construir o docker-compose que irá construir a nossa aplicação e connectar os *containers* que precisam de comunicar entre si.

No desenvolvimento do docker compose são criados 3 containers de forma a que um contenha a imagem do serviço de email, outro com a imagem do serviço de autorização e por fim um com a imagem do servidor de emails. Também é criado um volume que contenha a imagem *mongo* de forma a guardar a informação necessária que, também é um container, só que persiste a informação para que possa ser novamente utilizada. Por fim e de forma a que os *containers* comuniquem entre si é necessário definir três *networks* para estabelecer a comunicação entre: volume da base de dados e serviço de autorização; serviço de autorização e serviço de email; serviço de email e servidor de email.

4 Implementação

Nesta secção, pretendemos abordar em concreto a forma como está implementada toda a aplicação, com especial enfâse nas componentes de autenticação e email, visto que as imagens criadas a partir dessas são as que efetivamente foram criadas por nós. Além disso, pretendemos também abordar a construção que é feita pelo *Docker* apartir do ficheiro *docker-compose* por nós definido, bem como as *Dockerfiles* que criam as imagens a partir das componentes previamente referidas.

4.1 Serviço de Email

O serviço de email construído está implementado de tal forma que, quando um utilizador tenta aceder ao mesmo através da porta 3000, ele verifica se no *url* vem indicado o *token* de acesso ao serviço, ou se o utilizador já possui um token válido guardado como uma *cookie*. De seguida, segue o seguinte algoritmo:

- Se qualquer um dos casos se verificar, o serviço de email faz um pedido ao serviço de autenticação através da rede interna que liga ambos (*Network 2*) para que verifique se o token é válido para aquele utilizador.

  - Se o token é válido, o serviço de autenticação responde afirmadamente ao serviço de email através da rede interna e, finalmente o serviço de email permite o envio de email ao utilizador.
  - Se o token não for válido, o serviço de autenticação responde negativamente ao serviço de email através da rede interna e, nesse caso, o serviço de email redireciona o utilizador anónimo para o serviço de autenticação para que obtenha um token de acesso.

- Se nenhum dos casos se verificar, o serviço de email redireciona o utilizador anónimo para o serviço de autenticação para que obtenha um token de acesso.

Finalmente e, após o utilizador escrever quem deverá ser o recetor do email, bem como o assunto e a mensagem, o serviço de email comunica com o servidor de email através da rede interna que liga ambos (*Network 3*) pedindo-lhe que envie um email para o recetor indicado pelo utilizador com o assunto e mensagem também indicados pelo mesmo, onde o emissor do email é o **no-reply@vr-5.gcom.di.uminho.pt**.

4.2 Serviço de Autorização

O serviço de autorização está implementado de tal forma que, quando um utilizador acede ao mesmo através da porta 3001, este permite que o utilizador peça um token de acesso, ou que se registe, caso ainda não o tenha feito.

Ao registar um utilizador na base de dados,são guardados os seguintes dados na mesma:

- email.
- nome.
- hash da password.

Quando o utilizador pede um token de acesso, o serviço de autorização procura pelo email correspondente e produz um token com validade de 1 hora, caso o utilizador esteja registado. Finalmente guarda o token como uma *cookie* e redireciona o utilizador para o serviço de email , inserindo no *url* o token de acesso.

Além disso, a qualquer altura, o servidor de autorização está à escuta de pedidos de verificação de token e, quando recebe um, verifica que o token está de acordo com os parâmetros de geração de *tokens* da aplicação e que ainda se encontra válido, respondendo afirmativamente ou negativamente conforme o caso.

Os tokens de acesso são gerados através do módulo *jsonwebtoken* existente para *NodeJS*.

4.3 Docker-compose

O ficheiro *docker-compose.yml* está definido para criar 4 containers:

- **db**, que irá conter uma imagem da base de dados *MongoDB*.
- **servicoauth**, que irá conter uma imagem do serviço de autorização, criada como uma build automática no *dockerhub*.
- **servicoemail**, que irá conter uma imagem do serviço de email, criada como uma build automática no *dockerhub*.

- **servidoremail**, que irá conter a imagem *namshi/smtp*.

Antes de definir detalhadamente os *containers* é também importante referir que ainda são definidas três *Networks* (1,2 e 3) e também é definido um volume (*data-mongodb* com *driver* local).

O container da base de dados é construído com as seguintes características:

- **image** : *mongo:3.0*.
- **volumes**: *data-mongodb:/data/db*, visto que os dados das base de dados do *MongoDB* são sempre armazenados na diretoria */data/db*.
- **Networks**: *network 1*.

A porta do container da base de dados utilizada é a *default*: 27017, apenas acessível na rede interna do *Docker*, por um container que lhe tenha acesso.

O container do serviço de autorização é construído com as seguintes características:

- **image**: *nall1994/servicoauth*, que é a imagem construída pela build automática no *dockerhub* do serviço de autorização.
- **ports:** *3001:3001*, significando que esta aplicação escuta por pedidos na porta 3001 tanto no *host* como no *container*.
- **Networks:** *network1* e *network2*, ligando à rede interna com a base de dados e à rede interna com o serviço de email.

O container do serviço de email é construído com as seguintes características:

- **image:** *nall1994/servicoemail*, que é a imagem construída pela build automática no *dockerhub*, do serviço de email.
- **ports:** *3000:3000*, significando que esta aplicação escuta por pedidos na porta 3000 tanto no *host* como no *container*.
- **Networks:** *network2* e *network3*, ligando à rede interna com o serviço de autorização e à rede interna com o servidor de email.

O container do servidor de email é construído com as seguintes características:

- **image:** *namshi/smtp*, imagem do servidor *SMTP*, responsável pelo envio do email.
- **networks:** *network3*, ligando à rede interna com o serviço de email.

Para finalizar, é também relevante referir que todas as *Networks* possuem a *driver bridge*.

4.4 Dockerfiles

4.4.1 Serviço de email

Esta imagem é construída a partir da imagem *node:10*, e a aplicação *node* é instalada na diretoria */home/node/app*. São atribuidas permissões iguais às existentes na pasta do *host* à diretoria de trabalho para o utilizador *node*. De seguida, são copiados os ficheiros de *package*

do *node* e é executado o comando *npm install* para que instale todos os pacotes necessários. De seguida, são copiados os conteúdos (código fonte) para a diretoria de trabalho exceto as pastas e ficheiros referidos no *.dockerignore* e são atribuídas as permissões ao utilizador *node* para acesso a essas pastas. É exposta a porta 3000 e, para posteriormente iniciar a imagem no container, é definido o **CMD:** *npm start*.

4.4.2 Serviço de autorização

Esta imagem é construída a partir da imagem *node:10*, e a aplicação *node* é instalada na diretoria */home/node/app*. São atribuidas permissões iguais às existentes na pasta do *host* à diretoria de trabalho para o utilizador *node*. De seguida, são copiados os ficheiros de *package* do *node* e é executado o comando *npm install* para que instale todos os pacotes necessários. De seguida, são copiados os conteúdos (código fonte) para a diretoria de trabalho exceto as pastas e ficheiros referidos no *.dockerignore* e são atribuídas as permissões ao utilizador *node* para acesso a essas pastas. É exposta a porta 3001 e, para posteriormente iniciar a imagem no container, é definido o **CMD:** *npm start*.

5 Manual de Utilização

Este manual assume que já tem o *docker* e o *docker-compose* instalados.
Para utilizar esta aplicação, é necessário seguir os seguintes passos:

- Navegue até à diretoria onde se encontra o ficheiro *docker-compose.yml*.
- Execute o comando: *docker-compose up*.
- Abra um navegador e já pode utilizar a aplicação.

É relevante recordar que o serviço de email é acessível através da porta 3000 e o serviço de autorização é acessível através da porta 3001. Se não possui conta registada na aplicação, aceda ao serviço de autorização e registe-se. Finalmente, a qualquer serviço que aceda, o fluxo de execução da aplicação está completamente controlado para que tenha que seguir sempre os passos necessários para enviar o email:

- Pedir token de acesso, caso não tenha um válido.
- Enviar email.

6 Resultados e Discussão

Nesta secção, pretendemos demonstrar, através de imagens, a correcção da arquitetura da aplicação desenvolvida comparando com o que foi inicialmente pedido.

A partir da figura 3 é possível visualizar que o serviço de email tem conectividade com o servidor de email e com o serviço de autorização, mas não tem conectividade com a base de dados.

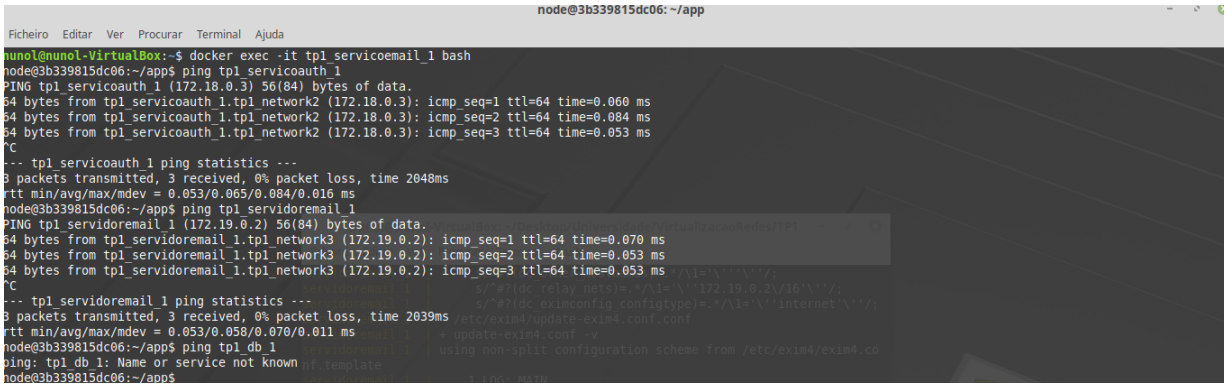

Figura 3: Sequência de comandos *ping* a partir do serviço de email

A partir da figura 4 é possível visualizar que o serviço de autorização tem conectividade com o serviço de email e com a base de dados, mas não tem conectividade com o servidor de email.

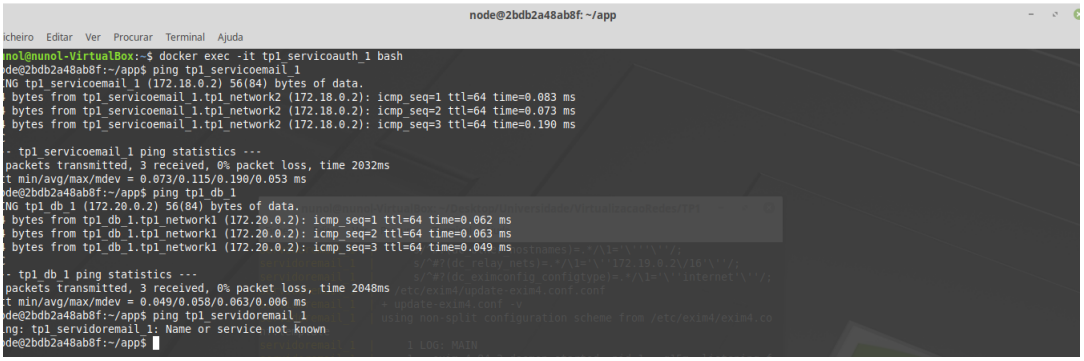

Figura 4: Sequência de comandos *ping* a partir do serviço de autorização

Na figura 5 é possível visualizar a página de pedido de token de acesso, após tentativa de acesso sem token ao serviço de email. É notória a indicação, através de uma mensagem, a necessidade de o utilizador se autenticar para ter acesso ao sistema.

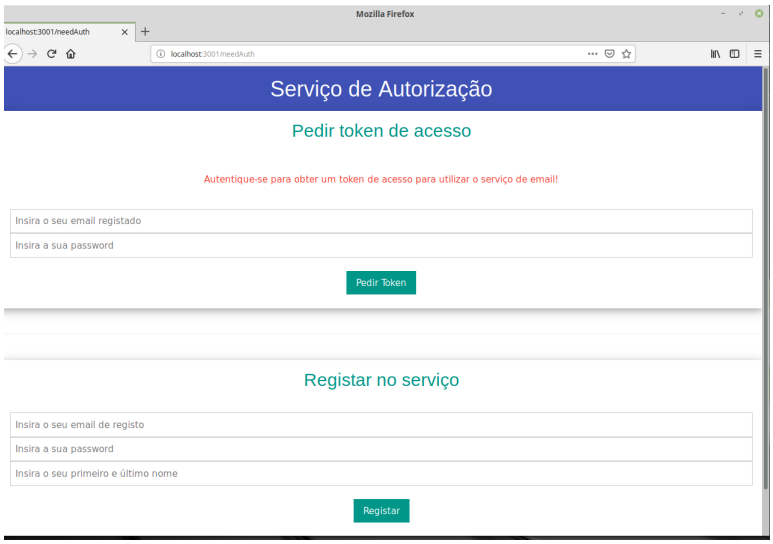

Figura 5: Página inicial do serviço de autorização após tentativa de acesso sem token

Na figura 6 é possível visualizar a página que permite o envio de um email, após ter confirmado a validade do token recebido seja por *url* ou pela *cookie* armazenada no *browser*.

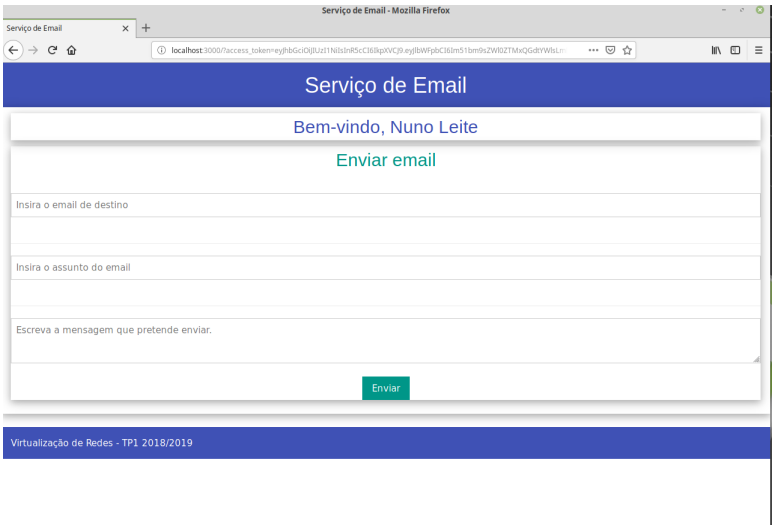

Figura 6: Página de envio de email

Estas quatro figuras são, na nossa opinião, as mais importantes para demonstrar que a arquitetura da aplicação desenvolvida está correta, visto que permite afirmar que a conetividade só existe entre os serviços necessários e que a necessidade de autorização através do token de acesso está sempre presente.

7 Conclusão

Na opinião do grupo, o resultado do trabalho realizado é muito satisfatório, visto que todos os objetivos principais foram atingidos. A comunicação entre aplicações está bem definida e restringida e, além disso, a parte funcional da aplicação (acesso através de *Oauth* e envio de email) também está bem realizada, apesar de que o envio de email é dependente da rede à qual se está ligado (Departamento de informática da universidade do minho).

As maiores dificuldades que o grupo encontrou resumem-se essencialmente aos extras pedidos, especificamente, à parte da construção de uma *proxy* que redirecionasse tráfego para ambas as aplicações. Existiu uma tentativa de aplicar este tipo de funcionamento à nossa arquitetura mas sem sucesso.

Em suma, no nosso ponto de vista, a resolução apresentada a este trabalho prático resolve todas as propostas inicialmente enunciadas, exceto os extras que poderiam ou não ser implementados, pelo que considerámos o resultado final bastante satisfatório.