Part I – Answers for Debug Systems Issues

1.
2.
    1.
    2.
3.
4.

Part II – Linux Laboratory

Criação da Máquina virtual, instalação e configuração do Sistema Operativo

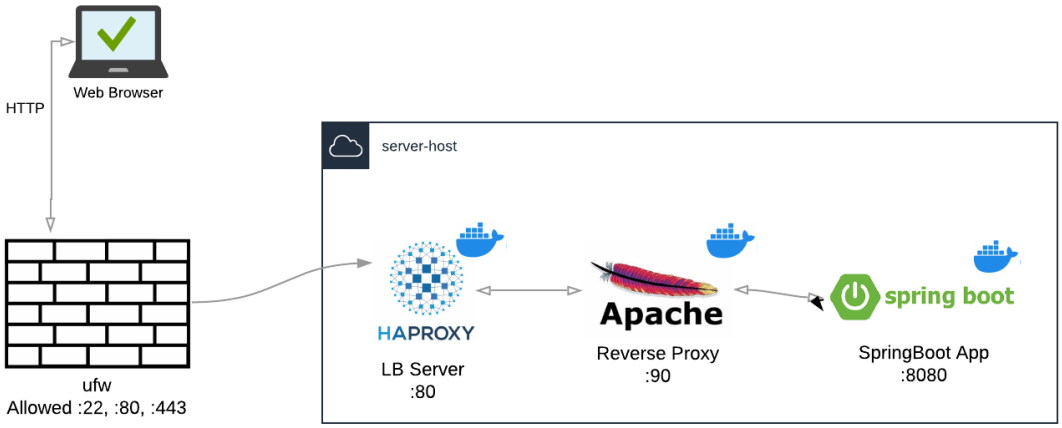
No meu caso, utilizei a versão 6.x do virtual box, de onde criei uma máquina virtual e instalei o SO conforme proposto.
Garanti que todos pacotes estão actualizados, instalei o *net-tools* , o *openssh* e configurei Bridged Adapter como o tipo de rede para a VM em causa.

> **NOTA:**
>
> > *Aspectos ligados a segurança como o caso de alteração da porta default ssh, criação de utilizadores com privilégios limitados, login por par de chaves RSA, etc, não foram levados em consideração assumindo que não é o que está sendo avaliado. (Mas reconhecendo a necessidade).*

A partir do comando `scp wit-cicd-challenge.jar wit@192.168.31.12:/home/wit/`, garanti que o ficheiro *.jar* fosse carregado da minha máquina (windows no meu caso) para a VM.

Intalação do docker e garantir que irá executar sem o *sudo*

Arquitetura e descrição da proposta



Criação e configuração da rede

Antes de iniciar com a criação dos containers, foi criada um rede bridge para conectarmos posterior conectar todos os containers que forem criados. O seguinte comando foi utilizado para criar a rede com o nome redewit

`docker network create --driver=bridge redewit`

Executando `docker network ls`, será possível confirmar a existência da rede previamente criada.

Criação e configuração do container SpringBoot

Por questões de organização, criaremos pastas para organizar os ficheiros relacionados à cada container. O container associado ao springBoot será denominado *wit-test*, pelo que a pasta criada poderá também ter o mesmo nome.

`cd ~`

`mkdir wit-test`

`cp wit-cicd-challenge.jar wit-test/`

`nano wit-test/Dockerfile`

Criei também na pasta o ficheiro com o nome **Dockerfile** e copiei o seguinte conteúdo:

```
FROM openjdk:11
COPY wit-test/wit-cicd-challenge.jar wit-cicd-challenge.jar
ENTRYPOINT ["java", "-jar", "/wit-cicd-challenge.jar"]
```

Onde:

- O **openjdk:11** é a imagem oficial criada pelo docker
- Na segunda linha a instrução **COPY** especifica que deverá ser copiado o ficheiro .jar
- Por fim, **ENTRYPOINT** especifica o comando a ser executado para o alojamento da aplicação quando for criado o conainer.

De seguida, executei o seguinte comando para criar uma imagem do Docker para o projeto Spring Boot atual:

`docker build -t wit-test wit-test/`

Nota que o primeiro primeiro parâmentro refere-se ao nome da imagem e o segundo à pasta onde deverá achar os ficheiros a serem usados para o build.

Finda a execução do último comando, poderá visualizar as imagens em causa a partir do comando `docker images`

Até então só existe a imagem que por sua vez está pronta para ser utilizada na criação do container. O seguinte comando criará o container, permitirá que esteja visível/acessível a partir de fora na porta 8080 e ainda garantirá que seja o serviço que iniciará com o sistema operativo:

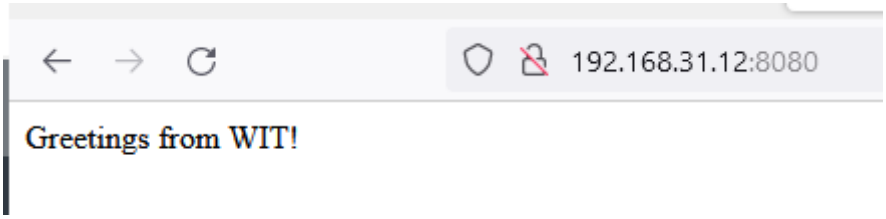
```
docker run -d --restart unless-stopped -p 8080:8080 --net redewit --name wit-test
```

É possível confirmar executando `docker ps` a existência do container e os respectivos detalhes.

Neste momento já podemos visualizar o resultado a partir do exterior (navegador da nossa máquina host que está rodar a VM):

> `<ip-do-seu-servidor>:8080`

O resultado deverá ser idêntico ao seguinte:



Criação e configuração do Reverse Proxy

Agora que configuramos o container da aplicação, partiremos para a configuração do reverse proxy que por sua vez fará o forward do tráfego para a aplicação.

Antes de tudo, utilizei o comando `docker pull httpd:latest` para puxar a imagem da última versão do httpd.

Na pasta proxy, criamos o directório **proxy** para conter o ficheiro **Dockerfile**

`mkdir proxy`

`nano proxy/Dockerfile`

Conteúdo do ficheiro:

```
# The Base Image used to create this Image
FROM httpd:latest

# to Copy a file named httpd.conf from present working directory to the
/usr/local/apache2/conf inside the container
COPY httpd.conf /usr/local/apache2/conf/httpd.conf

# This is the Additional Directory where we are going to keep our Virtualhost
configuraiton files
RUN mkdir -p /usr/local/apache2/conf/sites/

# To tell docker to expose this port
EXPOSE 90

CMD ["httpd", "-D", "FOREGROUND"]
```

Ainda na pasta criada **proxy**, criei o ficheiro de configuração `nano httpd.conf` com o conteúdo do seguinte link:

O Build e criação da imagem a partir do ficheiro **proxy/Dockerfile** será realizado após executar o comando `docker build -t proxy proxy/` e o mesmo é denominado *proxy* e pode ser confirmado executando `docker images`

Segue a criação do workspace que será usado para o *mount* no container e irá conter alguns ficheiros de configuração. São 2 directórios, onde o primeiro armazena os ficheiros *.conf* e o segundo os ficheiros *html*

`mkdir -p /home/wit/apps/docker/apacheconf/sites`

`mkdir -p /home/wit/apps/docker/apacheconf/htmlfiles`

Agora a criação do ficheiro *.conf* denominado *demowit* para conter o conteúdo a seguir:

`nano /home/wit/apps/docker/apacheconf/sites/demowit.conf`

O conteúdo:

```
<VirtualHost *:90>

    ServerName demowit.local
    ServerAlias www.demowit.local

    ServerAdmin demo@exemplo.com
    DocumentRoot /usr/local/apache2/demowit

    <Directory "/usr/local/apache2/demowit">
        Order allow,deny
        AllowOverride All
        Allow from all
        Require all granted
    </Directory>

    #Load the SSL module that is needed to terminate SSL on Apache
    LoadModule ssl_module modules/mod_ssl.so

    #This directive toggles the usage of the SSL/TLS Protocol Engine for proxy.
```

```
    #Without this you cannot use HTTPS URL as your Origin Server
    SSLProxyEngine on

    # To prevent SSL Offloading
    # Set the X-Forwarded-Proto to be https for your Origin Server to understand
that this request is made over HTTPS

    RequestHeader set X-Forwarded-Proto “https”
    RequestHeader set X-Forwarded-Port “443”

    ErrorLog logs/demowit-error.log
    CustomLog logs/demowit-access.log combined

    # The ProxyPass directive specifies the mapping of incoming requests to the
backend server (or a cluster of servers known as a Balancer group).
    # It proxies the requests only with matching URI “/blog”

    #To ensure that and Location: headers generated from the backend are modified
to point to the reverse proxy, instead of back to itself, #the ProxyPassReverse
directive is most often required:

    ProxyPass /wit-test http://127.0.0.1:8080/
    ProxyPassReverse /wit-test http://127.0.0.1:8080/

</VirtualHost>
```

Agora a criação do ficheiro html que servirá para landing page

`nano /home/wit/apps/docker/apacheconf/htmlfiles/index.html`

O conteúdo

```
<html>
    <head>
        <title>demowit</title>
    </head>
    <body>
        <h2> It's working... </h2>
    </body>
</html>
```

A última configuração para esta etapa é a criação do container, associado à publicação da porta e o mount dos directórios/ficheiros a serem utilizados no container.

`docker container run`
`--publish 90:90`
`-d --restart unless-stopped`
`--name proxy --net redewit`
`-v /home/wit/apps/docker/apacheconf/sites:/usr/local/apache2/conf/sites`
`-v /home/wit/apps/docker/apacheconf/htmlfiles:/usr/local/apache2/demowit`
`proxy`

Lembrando que configurei a porta 90 para o reverse proxy.

No ficheiro */etc/hosts*, deve associar o ip ao dns local que está sendo utilizado nos ficheiros:

- 127.0.0.1 demowit.local

> Para testar esta configuração:
>
> > `<ip-do-seu-servidor>:90` no browser, fora do server e mesma rede, e `curl demowit.local:90/wit-test/` dentro do server

- (Aqui entram
    - imagens de demonstração,
        - ambos cenários)

Criação e configuração do LB

Configurações gerais do processo

Agora que o fluxo todo está funcionar, activaremos o firewall para garantir que os acessos serão apenas a partir da porta **:80** para http e porta **:22** para conectar ao server usando SSH.

```
sudo ufw limit 22/tcp
sudo ufw allow http
sudo ufw allow https
sudo ufw enable
```

Conclusão