Universidade do Estado de Santa Catarina – Bacharelado em Ciência da Computação
TE em Ciência da Computação – Prof. Rafael R. Obelheiro

Exercícios – Docker

Muitos dos comandos usados para resolver os exercícios a seguir exigem privilégio de `root`.

1. Usando o APT, instale o pacote `docker.io`.
2. Para testar sua instalação de Docker, tente baixar e executar a imagem `hello-world`:

   ```
   $ sudo docker run hello-world
   ```

3. Use `docker images` para descobrir o tamanho da imagem `hello-world`.
4. Baixe a imagem mais recente de Ubuntu (`ubuntu`) usando `docker pull`. Qual o tamanho dessa imagem?
5. Execute a imagem lançando um shell:

   ```
   $ sudo docker run -it ubuntu bash
   ```

   Use `top` para descobrir quantos processos estão executando no contêiner, e quais são eles.
6. Em uma outra janela, execute novamente a imagem usando o mesmo comando do exercício 5. A seguir, crie um arquivo dentro do contêiner, verificando que ele foi criado com sucesso e possui o conteúdo desejado. Volte para a janela anterior e verifique se o arquivo criado no segundo contêiner está visível no primeiro.
7. Retorne à segunda janela (sessão onde o arquivo foi criado), e use o comando `exit` para encerrar o shell. Use `docker ps` para verificar se os dois contêineres continuam executando.
8. No contêiner criado no exercício 5, use APT para instalar o pacote `mtr-tiny` (caso o pacote não seja encontrado, atualize a lista de pacotes e tente novamente).

   O comando `mtr` mostra a rota percorrida por pacotes de rede até um determinado nó. Execute o comando `mtr 1.1.1.1` para ver a rota entre o contêiner e o servidor DNS da Cloudflare. Deixe o comando executando enquanto continua os exercícios.
9. Em uma outra janela, use `docker exec` para abrir um shell no contêiner com `mtr`. Dentro do contêiner, use `ps` ou `top` para ver os processos que estão executando. Em outra janela (fora do contêiner), use `docker top` para ver os processos em execução no contêiner, e compare com a lista obtida dentro do contêiner.
10. Use o comando `docker stop` para parar o contêiner. A seguir, use o comando `docker ps` para se certificar que ele foi encerrado.
11. Execute novamente a imagem, e veja se o `mtr` continua disponível no contêiner.
12. Os arquivos de aula contêm uma aplicação exemplo em Flask, um *framework web* para Python. Neste exercício, você irá configurar de forma interativa um contêiner para executar a aplicação.
    (a) Descompate os arquivos de aula, e entre no diretório `flask-megasena`.
    (b) Usando o Docker, execute a imagem `alpine:3.5`, abrindo um shell. (A imagem `alpine` não tem `bash`, apenas `sh`.)
    (c) No contêiner, execute os comandos (`#` é o prompt do shell):

    ```
    # apk add --update py2-flask
    # mkdir -p /usr/src/app/templates
    ```

(d) Fora do contêiner, copie os arquivos da aplicação (`12ab` é o ID do contêiner):

```
$ sudo docker cp app.py 12ab:/usr/src/app/
$ sudo docker cp templates/index.html 12ab:/usr/src/app/templates
```

(e) Fora do contêiner, salve a imagem com o programa:

```
$ sudo docker commit -m "Megasena no Alpine" 12ab megasena-alpine
```

(f) Verifique que a imagem foi salva, e qual o seu tamanho.

(g) Encerre a execução do contêiner.

(h) Use o comando abaixo para instanciar a imagem salva e executar a aplicação no contêiner:

```
$ sudo docker run -p 7000:5000 megasena-alpine python /usr/src/app/app.py
```

(i) Abra o navegador e acesse a URL `localhost:7000` para ver os números da Mega-Sena.

(j) Encerre a execução do contêiner.

13. Neste exercício, você irá usar um Dockerfile para automatizar a criação e configuração do contêiner do exercício 12.

(a) Use um editor de texto para criar o arquivo `Dockerfile` com o conteúdo abaixo:

```
# Dockerfile para o exercicio
# imagem base
FROM alpine:3.5

# instala Python e Flask
RUN apk add --update py2-flask

# copia os arquivos exigidos pela aplicacao
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# porta de rede usada pelo container
EXPOSE 5000

# executa a aplicacao
CMD ["python", "/usr/src/app/app.py"]
```

(b) Crie a imagem usando o comando:

```
$ sudo docker build -t "megasena:dockerfile" .
```

(c) Instancie a nova imagem:

```
$ sudo docker run -d -p 7000:5000 megasena:dockerfile
```

(d) Abra o navegador e acesse a URL `localhost:7000` para ver os números da Mega-Sena.

14. Se você inspecionar o código da aplicação em `app.py`, verá que ela registra um log dos números sorteados em `/var/megasena`, mas esses logs são perdidos a cada vez que a imagem é instanciada. Uma forma de contornar isso é usando um volume:

```
$ sudo docker volume create mega-logs
$ sudo docker run -v mega-logs:/var/megasena -p 7000:5000 megasena:dockerfile
```

Fora do contêiner, use `docker volume inspect` para descobrir onde estão são armazenados os logs, e comandos como `ls` e `cat` para examiná-los.

15. Execute a imagem `alpine:3.5` em *background*, com o comando `ping 8.8.8.8`. Use `docker logs` para examinar e monitorar os logs do contêiner.

16. Os comandos `docker stop` e `docker pause` permitem, respectivamente, parar e pausar um contêiner. Um contêiner parado por ser reiniciado com `docker restart`, e um contêiner pausado com `docker unpause`. Analise os logs do contêiner do exercício anterior quando são usados esses pares de comandos (`stop`/`restart` e `pause`/`unpause`), e determine a diferença entre eles.

17. Encerre e remova todos os contêineres que porventura estejam executando ou não tenham sido removidos.