Introdução ao Docker

Jean Phelipe de Oliveira Lima

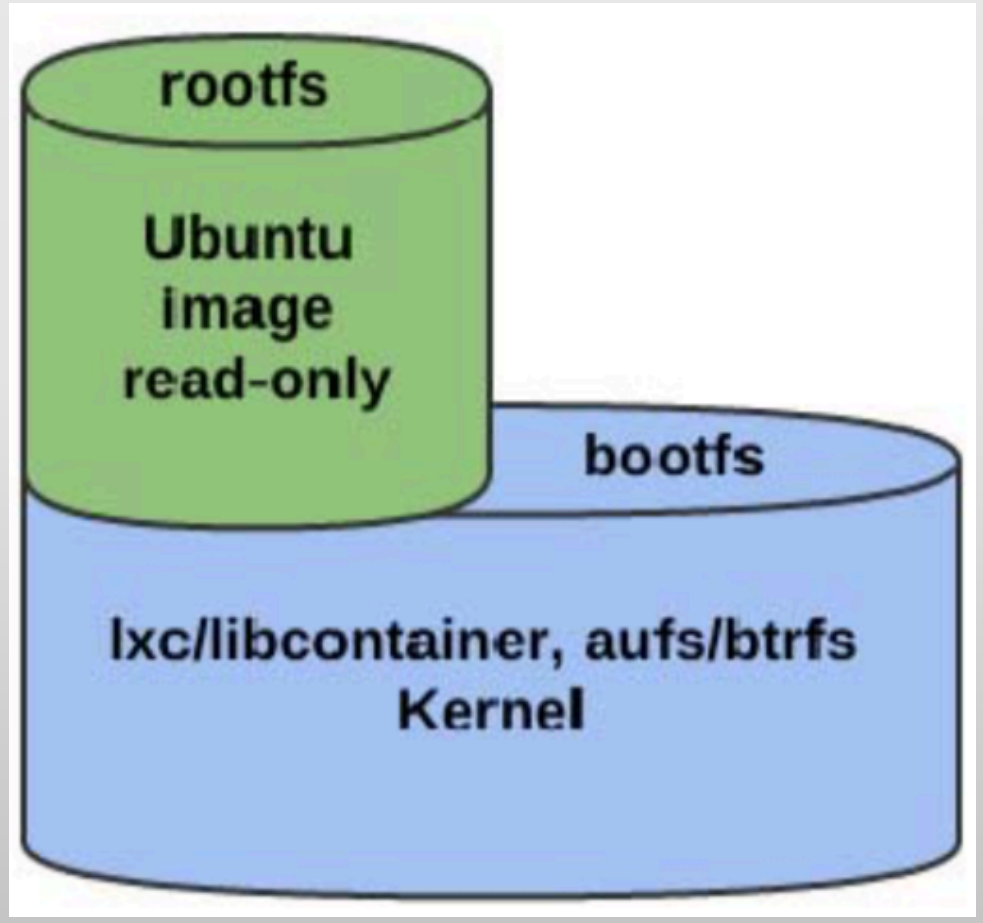
jpdol.eng16@uea.edu.br

/jpdol

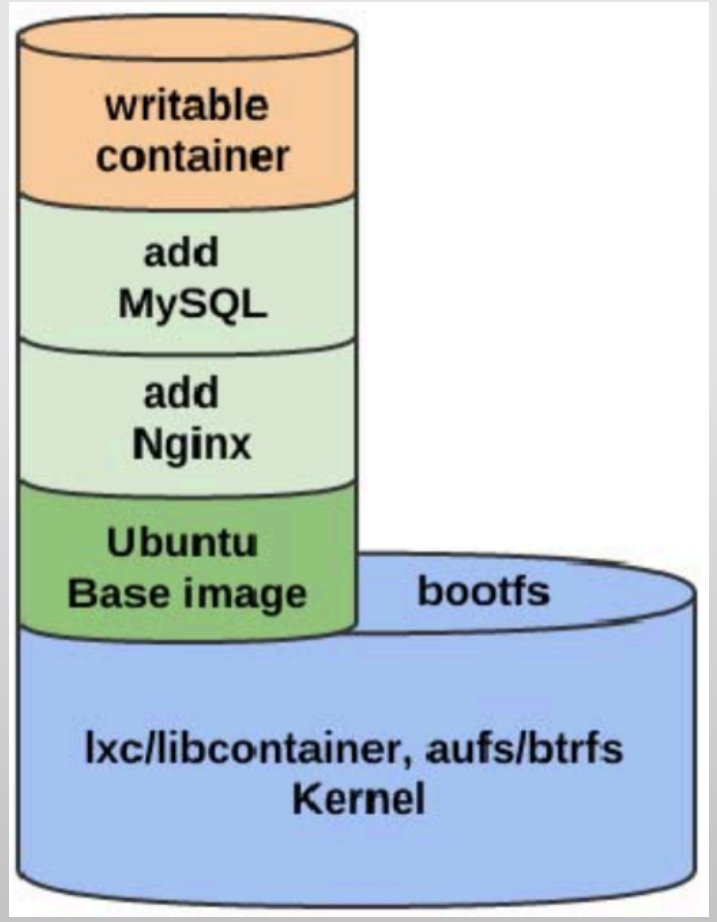
Construção de Imagens

- Funcionamento
- DockerFile
- Mapeamento de Gates
- Cópia de Arquivos
- Definição de Diretório de Trabalho
- Inicialização de Serviços
- Tratamento de Logs
- Exportação e Importação de Containers

Funcionamento



Funcionamento



Explorando o DockerFile

- Recurso feito para automatizar o processo de execução de tarefas no Docker.
- Com este recurso, podemos descrever o que queremos inserir em nossa imagem
- Desta forma, quando geramos um *build*, o Docker cria um *snapshot* com toda a instalação que elaboramos no *Dockerfile*.

- Resumo: O *Dockerfile* é um arquivo que aceita rotinas em *shell script* para serem executadas.

Explorando o DockerFile

- Exemplo Nginx:

```
FROM ubuntu
MAINTAINER <Seu nome> <seu@email.com>
RUN apt-get update
RUN apt-get install -y nginx
```

FROM – estamos escolhendo a imagem base para criar o *container*;

MAINTAINER – especifica o nome de quem vai manter a imagem;

RUN – permite a execução de um comando no *container*.

Explorando o DockerFile

- Build:

$ docker build -t nginx .

- Criar *container* para executar a aplicação:

$ docker run -d -p 8080:80 nginx /usr/sbin/nginx –g "daemon off;"

- *Request* para teste:

$ curl -IL http://localhost:8080
(ou, no caso do windows, $ curl -IL http://<IP-VM>:8080)

Explorando o DockerFile

- Até agora, utilizou-se para criar a imagem utilizada pelo *container* que executa o *web server* um Dockerfile, que consiste em um arquivo contendo diretivas de execução e instruções que servirão para criar a imagem.
- Os próximos passos serão par incrementar o Dockerfile com novas diretivas para otimizar o processo de construção da imagem e também os comandos de execução da aplicação.

Mapeando portas

- O comando para execução de um container com Nginx (docker run -d -p 8080:80 nginx /usr/sbin/nginx –g "daemon off;") está um pouco grande. Vamos reduzir a instrução de inicialização inserindo a opção **EXPOSE** no nosso *Dockerfile*. Além disso, faremos uso de um atalho para mapeamento de portas, o parâmetro **-P**.

- No *Dockerfile*, adicionamos a instrução **EXPOSE**:

```
FROM ubuntu
MAINTAINER <Seu Nome> <Seu@Nome.com>
RUN apt-get update
RUN apt-get install -y nginx
EXPOSE 80
```

Mapeando portas

- Em seguida, nós atualizamos a imagem executando o **build** novamente:

```
$ docker build -t nginx .
```

- O Docker apenas atualizou a imagem que já existia, acrescentando a instrução para expor a porta 80 sempre que um novo container for criado.

- Agora, podemos testar criando uma nova instância:

```
$ docker run -d -P nginx /usr/sbin/nginx -g "daemon off;"
```

Mapeando portas

- Ao criar um container passando a instrução **-P**, estamos permitindo que o Docker faça o mapeamento de qualquer porta utilizada no *container*, para alguma outra porta no *host*. Conferiremos isso com o seguinte atalho:

$ docker port <ContainerID>

- A saída será algo do tipo:

→ 80/tcp -> 0.0.0.0:41234

onde 41234 é a porta escolhida pelo Docker.

Mapeando portas

- Sempre que um *container* for criado desta forma, o retorno da instrução **-P** será uma porta aleatória criada no *host*. Se testarmos o acesso ao Nginx na porta criada através de request:

```
$ curl -IL http://localhost:41234/
```

teremos o seguinte retorno:

→ HTTP/1.1 200 OK
→ Server: nginx/1.4.6 (Ubuntu)

Cópia de Arquivos

- (Aqui está o detalhe oculto que não falei na aula, e vai ficar como desafio p vcs!)
- Podemos criar um *container*, acessar seu *shell*, sair e salvar as configurações (*commit*).
- o Nginx está configurado para utilizar a porta 80 (padrão para o protocolo TCP)
- Imagine se você precisar fazer várias alterações nos arquivos de configuração do *web server*, a produtividade pode ser comprometida. Para resolver isso, vamos utilizar a opção **ADD** no *Dockerfile*. Desta forma, vamos referenciar o arquivo que queremos copiar e o local de destino para a imagem durante o processo de build.

Cópia de Arquivos

- Para testar o **ADD**, vamos mudar um arquivo do Nginx, para mudar a sua porta padrão de 80 (padrão para o protocolo TCP), para 8080.

- O Dockerfile fica da seguinte maneira:

```
FROM ubuntu
MAINTAINER <Seu Nome> <Seu@nome.com>
RUN apt-get update
RUN apt-get install -y nginx
ADD exemplo /etc/nginx/sites-enabled/default
EXPOSE 8080
```

Cópia de Arquivos

- Com a instrução **ADD**, o arquivo chamado **exemplo** será copiado para o diretório **/etc/nginx/sites-enabled**, e será chamado de **default**. O arquivo exemplo deve existir no mesmo contexto do *Dockerfile*. O seu conteúdo é bem simples e foi alterado apenas para o Nginx utilizar a nova porta, no nosso caso, a 8080:

```
server {
        listen 8080 default_server;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html index.htm;
}
```

Cópia de Arquivos

- Agora, podemos executar o nosso processo de build e gerar a nova imagem, e o arquivo será copiado para as configurações do Nginx. Em seguida vamos executá-lo e realizar *request*:

```
$ docker build -t nginx .
```

```
$ sudo docker run -d nginx /usr/sbin/nginx -g "daemon off;"
```

```
$ curl -IL http://localhost:8080
```

Cópia de Arquivos

- Você verá que aconteceu um erro. Isso ocorre porque a partir de agora, não existe a mais a conversão da porta 80 (em que a mensagem era escrita), para a porta 8080 (de onde a mensagem era lida). Neste momento a mensagem está na 8080 do *container*, por isso deve ser acessada da seguinte forma:

  - Primeiro, descobrimos o IP do *container*:

```
$ docker inspect aeaeaeaeaeae | grep IPAddress
ou
$ docker exec -it <ContainerID> ifconfig
```

  - Como mensagem de retorno encontraremos algo como:

    - → "IPAddress": "192.180.0.1"

Cópia de Arquivos

- Já temos o IP que está sendo utilizado, então podemos refazer o teste de *request* na porta 8080, e verificar o funcionamento:

```
curl -IL http://192.180.0.1:8080
```

Agora o Nginx deve estar funcionando.

Definição de Área de Trabalho

- A área de trabalho padrão do Docker é o diretório raiz /. Podemos alterar isso durante a criação de um *container* ao usarmos a opção **-w**, ou tornando padrão usando a diretiva **WORKDIR** no *Dockerfile*.

- OBS: A utilização dessa diretiva é muito específica para cada aplicação no mundo real. Muitas vezes não se utiliza. Para continuar com o exemplo do Nginx, não modificaremos a área de trabalho.

Inicialização de Serviços

- O comando utilizado para geração de containers ficou bem menor. Entretanto, podemos reduzi-lo um pouco mais, informando no *Dockerfile* a instrução **CMD**, para que ele possa inicializar o Nginx sempre que um container novo for criado. A seguir o Dockerfile atualizado:

```
FROM ubuntu
MAINTAINER <Seu nome> <seu@nome.com>

RUN apt-get update
RUN apt-get install -y nginx
ADD exemplo /etc/nginx/sites-enabled/default

RUN echo "daemon off;" >> /etc/nginx/nginx.conf

EXPOSE 8080
CMD service nginx start
```

Inicialização de Serviços

- Recrie a imagem, inicialize um *container* e faça o teste de *request*:

```
$ docker build -t nginx .

$ docker run -d nginx

$ curl -IL http://192.180.0.1:8080
```

Tratamento de Logs

- O Docker possui um recurso para visualizar os *logs* de saída (**stdout**) e de erro padrão (**stderr**). Isso é interessante para verificarmos o que está acontecendo dentro de um *container*, sem a necessidade de conferir um determinado arquivo de *log*.

- Para este exemplo, vamos utilizar o *Dockerfile* da seção anterior e redirecionar os *logs* do Nginx para **stdout** e **stderr**.

Tratamento de Logs

- O *Dockerfile* ficará assim:

```
FROM ubuntu
MAINTAINER <Seu Nome> <seu@nome.com>
RUN apt-get update
RUN apt-get install -y nginx
ADD exemplo /etc/nginx/sites-enabled/default
RUN ln -sf /dev/stdout /var/log/nginx/access.log
RUN ln -sf /dev/stderr /var/log/nginx/error.log
EXPOSE 8080
CMD ["nginx", "-g",  "daemon  off;"]
```

Tratamento de Logs

- Recrie a imagem com **build** e, em seguida, um container:

```
$ docker build -t nginx .
```

```
$ docker run -d -p 80:80 nginx
```

- Em seguida faça alguns *requests*:

```
$ for ((i=1; i<=10; i++)); do curl -IL http://127.0.0.1:8080; done
```

Tratamento de Logs

- Agora, podemos conferir os logs utilizando o comando **docker logs** e a identificação do nosso container:

$ docker logs <ContainerID>

- Desta forma, conseguimos capturar os *logs* do serviço Nginx de dentro do *container* sem precisar abrir os arquivos de *log*, pois todo o *log* está sendo redirecionado de forma que a opção **logs** do Docker consiga acompanhar.

Exportação e Importação de Containers

- Podemos criar uma imagem partindo de um *container* que está funcionando, e gerar um arquivo .tar usando a opção **save**.
- Para criar a nova imagem, basta fazer um *commit* no *container* em execução:

$ docker commit <ContainerID> <Nova Imagem>

- De posse da nova imagem que foi gerada, faremos a exportação criando um arquivo:

$ docker save <Nova Imagem> > /tmp/<Nova Imagem>.tar

Exportação e Importação de Containers

- Agora o arquivo .tar que foi criado pode ser enviado para o outro *host*. Para fazer a importação desse arquivo, utilizamos a opção **load**:

$ docker load < /tmp/<Nova Imagem>.tar

Introdução ao Docker

Jean Phelipe de Oliveira Lima

jpdol.eng16@uea.edu.br

/jpdol