Jose Luiz Bento Da Silva / luizbsilva@gmail.com

4LINUX
OPEN SOFTWARE SPECIALISTS

CURSO 525

INFRAESTRUTURA ÁGIL COM PRÁTICAS DEVOPS

4

Docker Containers

Conceitos de Contêineres

Anotações

Objetivos da aula

1. Introdução ao Docker
2. Plataformas de Execução
3. Arquitetura
4. Funcionamento

Anotações

Introdução ao Docker

Definição

✔ O Docker é uma plataforma Open Source escrita em Go. Uma linguagem de programação de alto desempenho desenvolvida dentro do Google, que facilita a criação e administração de ambientes isolados.

✔ Descreve-se, como uma plataforma aberta para desenvolvedores e administradores de sistemas para construir, compartilhar e executar aplicações distribuídas. Ele permite que você execute suas aplicações em container.

Anotações

Introdução ao Docker

Virtual Machines vs Docker

Ambiente Docker

Container é um processo em área restrita, que executa um aplicativo e suas dependências no sistema operacional hospedeiro.
A aplicação dentro do container em si, é o único processo em execução que roda na máquina, permitindo ao servidor hospedeiro executar vários containers de forma independente. Este modo de trabalho, permite remover conflitos entre dependências e simplificar a implantação de toda a instalação e configuração de forma instantânea, comparado com o modelo tradicional de virtualização.

Plataformas de Execução

Plataformas

O Docker Community Edition pode ser executado tanto no sistema operacional MacOS X, quanto no Windows 10, caso você esteja usando o Docker para desenvolvimento, treinamento ou aprendizado de local. O Docker Enterprise Edition é executado em uma variedade de diferentes distribuições Linux e no Microsoft Windows Server 2016.

O Docker Enterprise Edition e o Community Edition podem ser executados na nuvem pública; E essa é uma das grandes vantagens do Docker, os contêineres que você cria e executa em seu sistema operacional de desktop local, podem ser facilmente movidos para a nuvem pública, ou em seu ambiente de servidor local que execute esses contêineres em produção.

Antes de instalar o Docker Enterprise, é preciso fazer um checklist nos seguintes itens:

Portas de rede: Uma série de portas precisam ser abertas para o tráfego de entrada;

Espaço em disco: É preciso 5GB de espaço livre em disco para a partição /var para nós manager (recomendado um mínimo de 6 GB). 500 MB de espaço livre para a partição /var para nós worker;

Sincronização de Horário: Em sistemas distribuídos, como o Docker UCP, a sincronização de tempo é fundamental para garantir a operação adequada. Como prática recomendada para garantir a consistência entre os mecanismos de um cluster UCP, todos os mecanismos devem sincronizar regularmente o tempo com um servidor NTP (Network Time Protocol);

Compatibilidade da versão do Docker Engine, DTR e UCP: O Docker EE é uma assinatura de software que inclui três produtos: Docker Engine com suporte de nível corporativo, Registro confiável do Docker(TR) e Docker Universal Control Plane(UCP).

Arquitetura

Docker Engine

Arquitetura

O Docker Engine é projetado como um aplicativo de servidor cliente, sendo composto de três coisas diferentes. Ele começa com o dockerd ou com o daemon do Docker, que é instalado quando você instala o Docker e esse servidor, que é o próprio servidor Docker.

Juntamente com a instalação do Docker Engine, você recebe uma API RESTful, ela é importante porque define a interface que todos os outros programas utilizam para falar com o daemon.

O cliente é o comando docker que você executa para conversar com o servidor Docker, para extrair imagens, criar imagens e instanciar contêineres. Independentemente da versão do Docker que você esteja usando, seja o Community Edition ou o Enterprise Edition, o Docker Engine é a base necessária para tornar tudo isso possível.

O daemon do Docker está instalado no host do Docker. Esse host do Docker pode ser seu computador desktop ou laptop, pode ser um servidor no datacenter ou pode ser uma máquina virtual sendo executada na nuvem. A partir daí, o host do Docker é usado para executar ou instanciar seus contêineres e imagens. É administrado através do cliente do Docker, que pode estar no mesmo host que o daemon do Docker ou pode ser remoto.

Funcionamento

Docker Namespace

✔ Os Namespaces asseguram que cada Container enxergue apenas o seu próprio ambiente, e não afete ou tenha acesso a processos em execução dentro de outros Containers.

✔ Além disso, os Namespaces fornecem acesso restrito aos sistemas de arquivos como, por exemplo, o Chroot por ter uma estrutura de diretório para cada Container.

Namespace

O Docker utiliza namespaces de processo, montagem, IPC, rede e usuário para isolar o que está acontecendo no host do Docker, com o que está acontecendo nos contêineres do Docker.
A Microsoft adicionou o equivalente a um isolamento de namespace no Windows, a fim de que o Docker para Windows forneça a mesma funcionalidade.
Os namespaces são semelhantes em conceito ao que um hipervisor faz para fornecer os recursos virtuais, como CPU virtual, memória virtual e armazenamento virtual à uma máquina virtual.
Os namespaces mantêm os contêineres isolados até os administradores do Docker, por exemplo, permitindo que os contêineres se comuniquem pelas redes virtuais do Docker no mesmo host.

Funcionamento

Funcionamento

Com o isolamento Namespace nos sistemas operacionais Docker, os aplicativos em execução nos contêineres parecem ter suas próprias árvores de processo, sistemas de arquivos, conexões de rede e muito mais. É possível ainda no Docker, mapear uma conta de usuário em um contêiner para uma conta de usuário no sistema operacional do host.

Alguns dos Namespaces que o Docker utiliza são:

- ✔ **PID**: O isolamento do processo (PID: ID do processo);
- ✔ **MNT**: Gerenciando Mount-ponts (MNT: Mount);
- ✔ **IPC**: Gerenciando o acesso a recursos IPC (IPC: Comunicação Inter Processo);
- ✔ **UTS**: Isolando o kernel e versão de identificadores (UTS: Unix timesharing sistema);
- ✔ **NET**: Interfaces de rede de gerenciando (NET: Networking).

Funcionamento

Docker Cgroups

✔ O Docker Engine em Linux, também faz uso de outra tecnologia chamada Cgroups ou grupos de controle. Com o uso de Cgroups, o Docker consegue isolar as aplicações e gerenciar apenas os recursos que você deseja.

✔ Os grupos de controle, permitem que o Docker compartilhe recursos de hardware disponíveis para Container, e se necessário, configure limites e restrições.

Cgroups

Em geral, os cgroups podem ser utilizados para controlar uma variedade de recursos, mas no Docker, os usos mais comuns são para limites de CPU de contêiner, reservas de CPU, limites de memória e reservas de memória.

Você deve saber que para usar cgroups no Docker, existem requisitos rígidos de kernel, então se você planeja usar cgroups, certifique-se de que seu kernel e sua versão do Linux serão compatíveis.

Funcionamento

Alguns dos Cgroups que o Docker utiliza são:

- ✔ **CPU**: Permite dar um peso proporcional aos Containers e, consequentemente, o recurso será compartilhado;
- ✔ **CPUSET**: Permite criar máscaras de CPU, executando threads dentro de um Container;
- ✔ **MEMORY**: Podemos definir limites de memória para um Container;
- ✔ **DEVICE**: Podemos definir quais dispositivos podem ser usados dentro do Container.

Recapitulando

1. Introdução ao Docker
2. Plataformas de Execução
3. Arquitetura
4. Funcionamento

Anotações

Docker Containers

Comandos essenciais do Docker

Anotações

Objetivos da aula

1. Comandos nativos
2. Docker Hub
3. Docker Images

Anotações

Comandos nativos

1 Conecte via SSH na máquina container e troque para o usuário root:

```
$ vagrant ssh container
$ sudo su -
```

2 Para visualizar informações do ambiente, execute o comando:

```
# docker system info
```

3 Para listar containers, imagens e redes no Docker, execute os comandos:

```
# docker container ls
# docker image ls
# docker network ls
```

container.4labs.example

Descrição dos comandos

- ✔ **docker info ou docker system info**: Exibe informações do Docker como versão, quantidade de containers em execução, storage drivers, entre outros;
- ✔ **docker ps ou docker container ls**: Lista containers em execução;
- ✔ **docker images ou docker image ls**: Lista imagens do Docker armazenadas localmente;
- ✔ **docker network ls**: Lista redes criadas para uso do Docker.

Comandos nativos

1 Para pesquisar por uma imagem, utilize o comando docker search:

```
# docker search debian
```

2 Para efetuar o download de uma imagem, utilize o comando docker image pull:

```
# docker image pull debian
```

3 Para executar um container, utilize o comando docker container run:

```
# docker container run -dit --name primeiro --hostname primeiro debian
```

container.4labs.example

Descrição dos comandos

- ✔ **docker search**: Pesquisa uma imagem no Docker HUB;
- ✔ **docker image pull**: Efetua o Download de uma imagem;
- ✔ **docker container run**: Executa a criação de um container;
  - ✔ **Opções**:
  - ✔ **-d**: Detached, executa o contêiner como um processo;
  - ✔ **-i**: Interactive, para poder interagir com o container;
  - ✔ **-t**:: Terminal, para que seja disponibilizado um TTY (Pseudo Terminal);
  - ✔ **--name**: Define o nome do container;
  - ✔ **--hostname**: Define o hostname do container.

Comandos nativos

1 Vamos listar os containers:

```
# docker container ls
```

2 Para se conectar ao container em execução, execute o seguinte comando:

```
# docker container attach primeiro
```

3 Execute alguns comandos no container:

```
$ ip -c a
$ hostname
$ cat /etc/hosts
$ exit
```

container.4labs.example

Anotações

Comandos nativos

1 Vamos listar os containers:

```
# docker container ls -a
```

Note que agora o container está parado, isto aconteceu pois ele recebeu um return code diferente de 0.

2 Inicie o container novamente e conecte-se ao mesmo:

```
# docker container start primeiro
# docker container attach primeiro
```

3 Utilize a sequência de teclas para desconectar do container, sem que ele seja parado:

```
<CTRL> + <P> <Q>
```

container.4labs.example

Descrição dos comandos

- ✔ **docker container ls -a:** Lista todos os containers (-a = all);
- ✔ **docker container start** ou **docker container stop:** Inicia ou pára um container;
- ✔ **<CTRL> + <P> <Q>** - Readescape sequence, comando utilizado para se desconectar de um container, sem parar o mesmo.

Comandos nativos

1 Vamos listar os containers:

```
# docker container ls -a
```

Note que agora o container ainda está em execução, isto ocorre devido a utilização do **Read escape Sequence**.

2 Para verificar os logs do container, utilize o comando:

```
# docker container logs primeiro
```

3 Pare e remova o container, após isto, verifique os containers existentes:

```
# docker container rm -f primeiro
# docker container ls -a
```

container.4labs.example

Descrição dos comandos

- ✔ **docker container logs:** Printa os logs na tela;
- ✔ **docker container rm -f:** Pára e remove o container(rm). A opção f é utilizada para que o container seja removido, mesmo se ele estiver em execução.

Comandos nativos

1 Vamos executar um container:

```
# docker container run -dit --name c1 --hostname c1 debian
# docker container ls
```

2 Vamos criar um arquivo e enviar para o container c1:

```
# echo "Arquivo de teste" > /tmp/arquivo
# docker container cp /tmp/arquivo c1:/tmp
```

3 Liste o arquivo e leia o conteúdo dentro do container:

```
# docker container exec c1 ls /tmp
# docker container exec c1 cat /tmp/arquivo
```

container.4labs.example

Descrição dos comandos

- ✔ **docker container cp:** Copia um arquivo para o container docker, também pode ser utilizado para retirar arquivos de um container;
- ✔ **docker container exec:** Executa um comando no container.

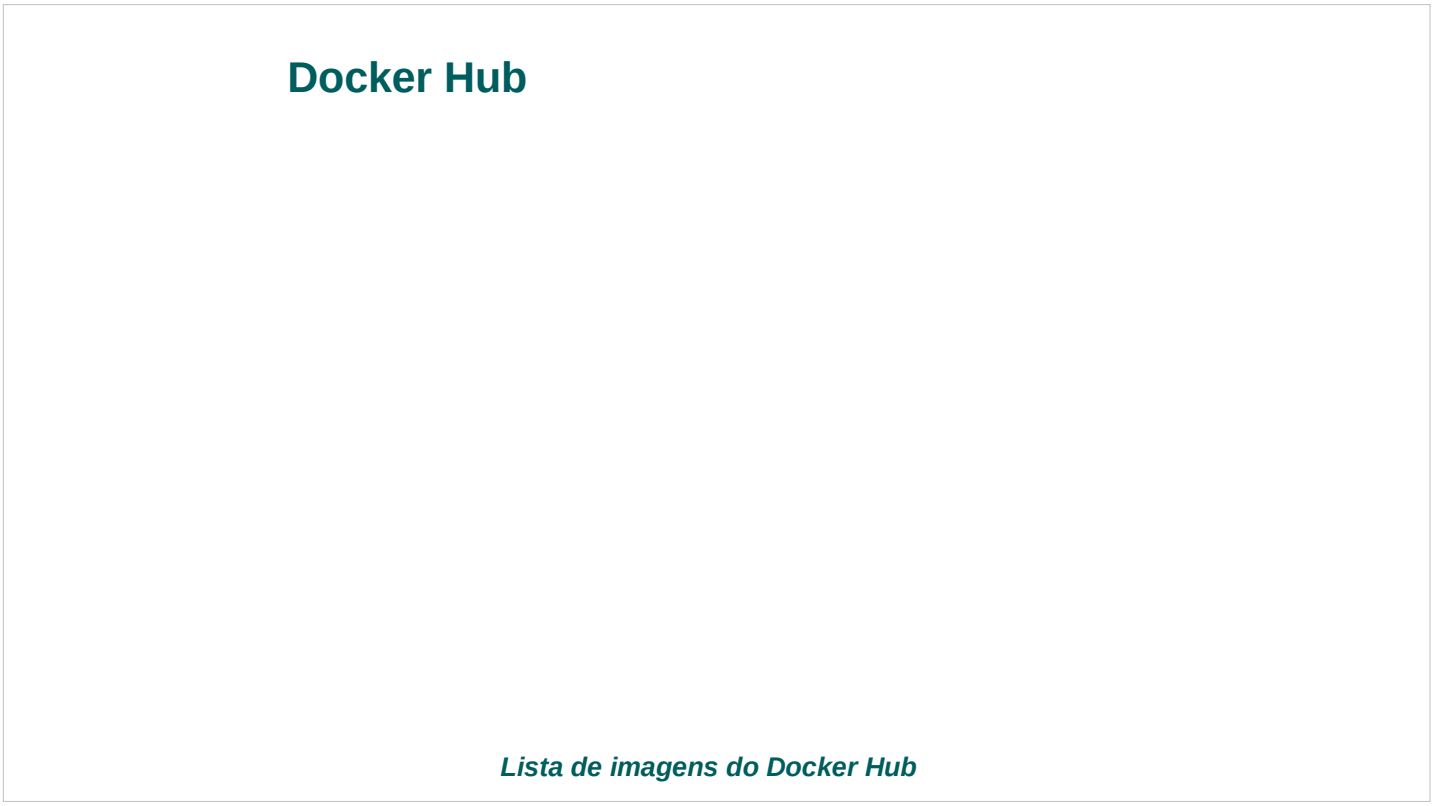
Docker Hub

✔ O Docker Hub é o maior repositório do mundo com imagens de contêiner e uma variedade de fontes de conteúdo, incluindo desenvolvedores de comunidades de contêineres, projetos de código aberto e fornecedores independentes de software (ISV), além de distribuir seu código em contêineres.

✔ Os usuários obtêm acesso a repositórios públicos gratuitos para armazenar e compartilhar imagens, ou podem escolher o plano de assinatura para repositórios privados.

Anotações

Docker Hub



Lista de imagens do Docker Hub

Docker HUB

O Docker Hub é um recurso centralizado para trabalhar com o Docker e seus componentes. Ele fornece os seguintes serviços:

- ✔ Hospedagem de imagens do Docker;
- ✔ Autenticação de usuário;
- ✔ Construções automáticas de imagens e ferramentas de fluxo de trabalho, como triggers de construção e ganchos da web;
- ✔ Integração com o GitHub e Bitbucket.

Criar conta no Docker HUB

Etapa 1

1 Acesse o endereço https://hub.docker.com/ e clique em **Sign Up** para criar uma conta:

Máquina Física

Anotações

Criar conta no Docker HUB

Etapa 2

2 Confirme o cadastro em seu e-mail e logue no Docker Hub, através do link **Sign In**:

Máquina Física

Anotações

Criar conta no Docker HUB

Etapa 3

3	Logue com o usuário e execute o comando para logar no Docker Hub: `$ docker login -u <usuario_docker_hub>` **<TECLE ENTER>** `Password:` **<digite a senha>**
4	Verifique se um arquivo de autorização foi criado: `$ cat .docker/config.json`
5	Para encerrar o login da conta do Docker Hub, execute o comando: `$ docker logout`

container.4labs.example

Descrição dos comandos

- ✔ **docker login:** Permite logar em uma conta no repositório Docker, sendo ele local ou remoto;
- ✔ **docker logout:** Permite encerrar o login em uma conta no repositório Docker, sendo ele local ou remoto.

Docker Images

Definição

✔ Uma imagem Docker, é um pacote executável que inclui tudo o que é necessário para executar um aplicativo, incluindo o código em tempo de execução, bibliotecas, variáveis de ambientes e arquivos de configuração.

✔ Já um contêiner, é uma instância de tempo de execução de uma imagem. Sendo assim, podemos resumir que uma imagem é um contêiner que ainda não está em execução.

Docker Images

As imagens do Docker possuem camadas intermediárias que aumentam a capacidade de reutilização, diminuem o uso do disco e aceleram a construção do docker, permitindo que cada etapa seja armazenada em cache. Essas camadas intermediárias não são mostradas por padrão.

Docker Images

Camadas em uma Imagem Docker

Docker Images

As imagens do Docker possuem várias camadas somente leitura e não podem ser modificadas. Você pode criar novas imagens a partir de imagens existentes, mas não pode modificar essas imagens existentes. Vários contêineres são geralmente baseados na mesma imagem, portanto, você pode usar essa imagem e gerar centenas ou milhares de contêineres.

Quando uma imagem é instanciada ou executada, como um contêiner, uma camada gravável superior é criada. Em seguida, essa camada gravável superior é realmente excluída quando o contêiner é removido. O Docker faz isso, usando drivers de armazenamento para gerenciar o conteúdo das camadas de imagem e a camada superior gravável.

Ao baixar uma imagem base do Docker, podemos adicionar a ela diversas camadas de leitura, antes da camada de gravação. Exemplo:

1. Vou baixar uma imagem base do Debian e adicionar uma aplicação como, por exemplo, um editor de textos;
2. A partir dessa alteração, crio uma nova imagem que vai conter as camadas Debian (Base Image) e emacs (Image);
3. Vou baixar uma imagem com a base do Debian + o editor de textos e adicionar uma segunda aplicação como, por exemplo, um servidor apache;
4. A partir dessa alteração, crio uma nova imagem que vai conter as camadas Debian (Base Image), emacs (Image) e Apache (Image);
5. Ao rodar um container usando a imagem com as três camadas, a última camada será de gravação, quando o container estiver em execução.

Docker Imagen

Contêiners e Camadas de Imagem

Docker Images

A principal diferença entre um contêiner e uma imagem, é a camada gravável superior. Todas as gravações no contêiner que adicionam novos dados ou modificam dados existentes, são armazenadas nessa camada gravável. Quando o contêiner é excluído, a camada gravável também é excluída. A imagem subjacente permanece inalterada.

Como cada contêiner tem sua própria camada de contêiner gravável e todas as alterações são armazenadas nessa camada de contêiner, vários contêineres podem compartilhar o acesso à mesma imagem subjacente e ainda ter seu próprio estado de dados. O diagrama acima mostra vários contêineres compartilhando a mesma imagem do Ubuntu 15.04.

Gerenciar Imagens no Docker

Comandos de Gerenciamento – ETAPA 1

1	Liste as imagens e verifique o histórico de comandos utilizados para sua construção: # docker image ls # docker image history debian
2	Para inspecionar uma imagem, utilizamos o seguinte comando: # docker image inspect debian
3	Antes de criar uma nova imagem, execute os seguintes comandos: # docker container run -dit --name servidor-debian debian # docker container exec servidor-debian apt update # docker container exec servidor-debian apt install apache2 -y

container.4labs.example

Gerenciando Imagens

- ✔ **docker image ls:** Lista imagens do Docker armazenadas localmente;
- ✔ **docker image history:** Mostra as camadas que compõe uma imagem;
- ✔ **docker image inspect:** Exibe informações detalhadas de um container ou imagem;
- ✔ **docker container exec:** Executa um comando de um container em execução.

Gerenciar Imagens no Docker

Comandos de Gerenciamento – ETAPA 2

4 Para criar uma nova imagem a partir das alterações feitas em um container, execute os seguintes comandos:

```
# docker container commit servidor-debian servidor-web
# docker image ls
```

5 Para inspecionar a nova imagem criada, a partir do container em execução:

```
# docker image inspect servidor-web
```

container.4labs.example

Gerenciando Imagens

✔ **docker commit:** Cria uma imagem, a partir de alterações realizadas em um container.

Gerenciar Imagens no Docker

Backup de Imagens

1

Salve a imagem servidor-web que inclui o apache para um arquivo tar:

```
# docker image save servidor-web > imagem-servidor-web.tar
# du -sh imagem-servidor-web.tar
```

2

Remova o container servidor-debian e apenas a imagem servidor-web:

```
# docker container rm -f servidor-debian
# docker image rm -f servidor-web
# docker image ls
```

container.4labs.example

Gerenciando Imagens

- ✔ **docker save:** Salva uma ou mais imagens em um arquivo tar;
- ✔ **docker image rm:** Remove uma imagem (a opção -f força a remoção).

Gerenciar Imagens no Docker

Restore de Imagens

1. Carregue a imagem servidor-web a partir do arquivo tar, e inicie o container servidor-web utilizando a nova imagem:

```
# docker image load < imagem-servidor-web.tar
# docker image ls
```

2. Execute o container com a imagem restaurada:

```
# docker container run -dit --name=servidor-web \
--hostname=servidor-web servidor-web
# docker container ls
```

container.4labs.example

Gerenciando Imagens

- ✔ **docker image load:** Carrega uma imagem do Docker, a partir de um arquivo tar.

Recapitulando

1. Comandos nativos
2. Docker Hub
3. Docker Images

Anotações

Docker Containers

Dockerfile

Anotações

Objetivos da aula

1. Dockerfile
2. Sintaxe
3. Criando o Primeiro Dockerfile
4. Dockerfile Servidor WEB

Anotações

Dockerfile

Definição

✔ O Docker pode criar imagens automaticamente, lendo as instruções de um Dockerfile. Dockerfile é um documento de texto que contém todos os comandos que um usuário pode chamar na linha de comando para montar uma imagem.

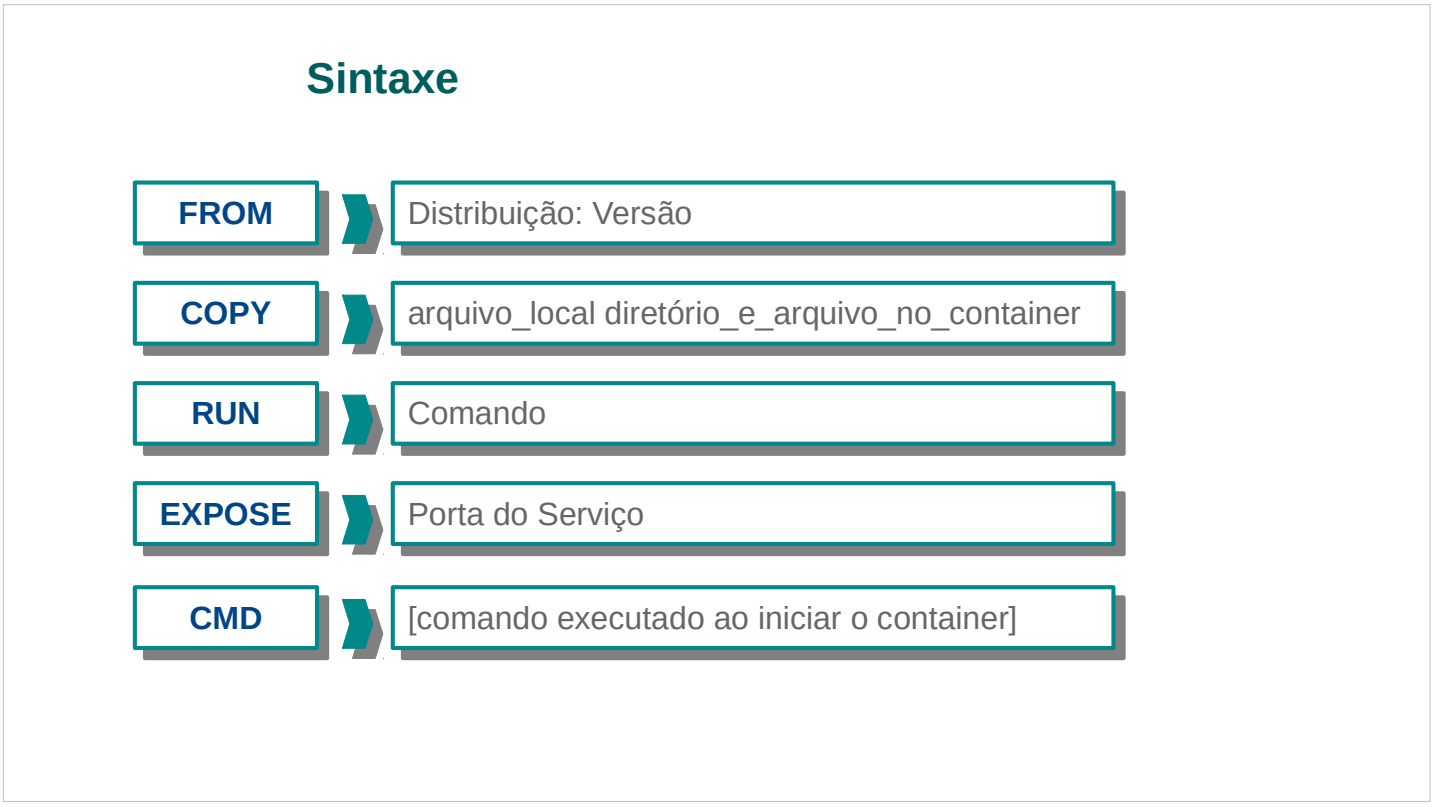
✔ O uso do comando docker build do Docker, pode criar uma compilação automatizada que executa várias instruções de linha de comando em sucessão.

Docker Images

O comando de compilação do docker cria uma imagem, a partir de um Dockerfile e de um contexto;
O contexto da construção é um conjunto de arquivos em um PATH ou URL de local especificado;
O PATH é um diretório em seu sistema de arquivos local. O URL é um local do repositório Git;
Um contexto é processado recursivamente. Portanto, um PATH inclui todos os subdiretórios e o URL inclui o repositório e seus submódulos.

Exemplo:

$ docker build .

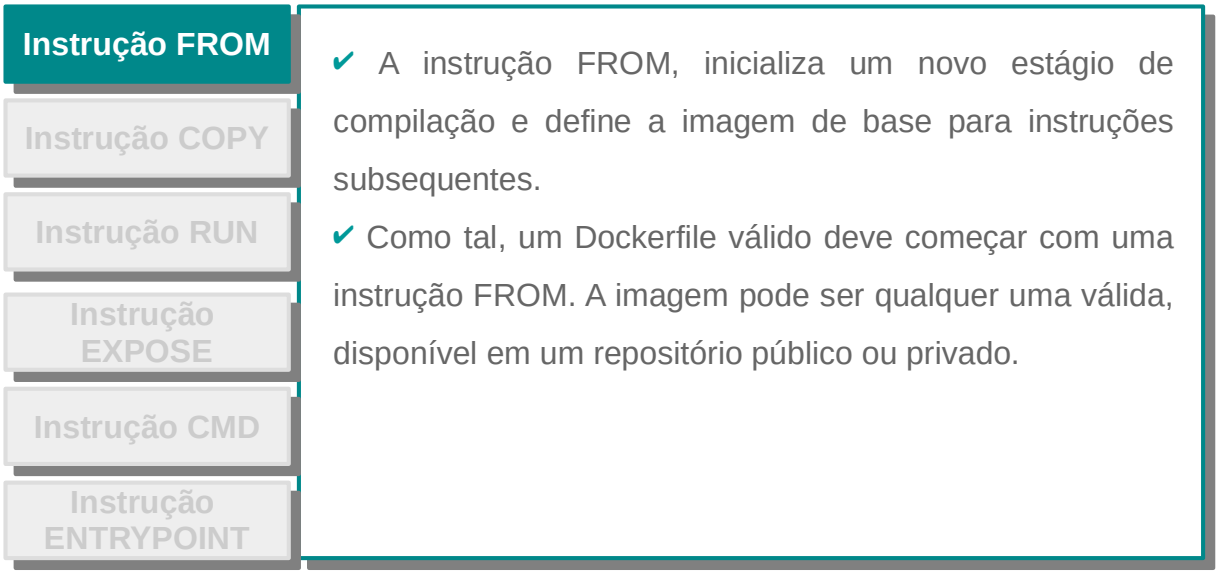


O arquivo de Dockerfile não faz distinção entre maiúsculas e minúsculas. No entanto, a convenção é que elas sejam MAIÚSCULAS para distingui-las dos argumentos mais facilmente.

O Docker executa instruções em um Dockerfile em ordem. Um Dockerfile deve começar com uma instrução `FROM`. A instrução FROM especifica a imagem base, a partir da qual você está construindo. FROM só pode ser precedido por uma ou mais instruções ARG que declaram argumentos que são usados em linhas FROM no Dockerfile.

O Docker trata as linhas que começam com # como um comentário, a menos que a linha seja uma diretiva de analisador válida. Um marcador # em qualquer outro lugar em uma linha, é tratado como um argumento.

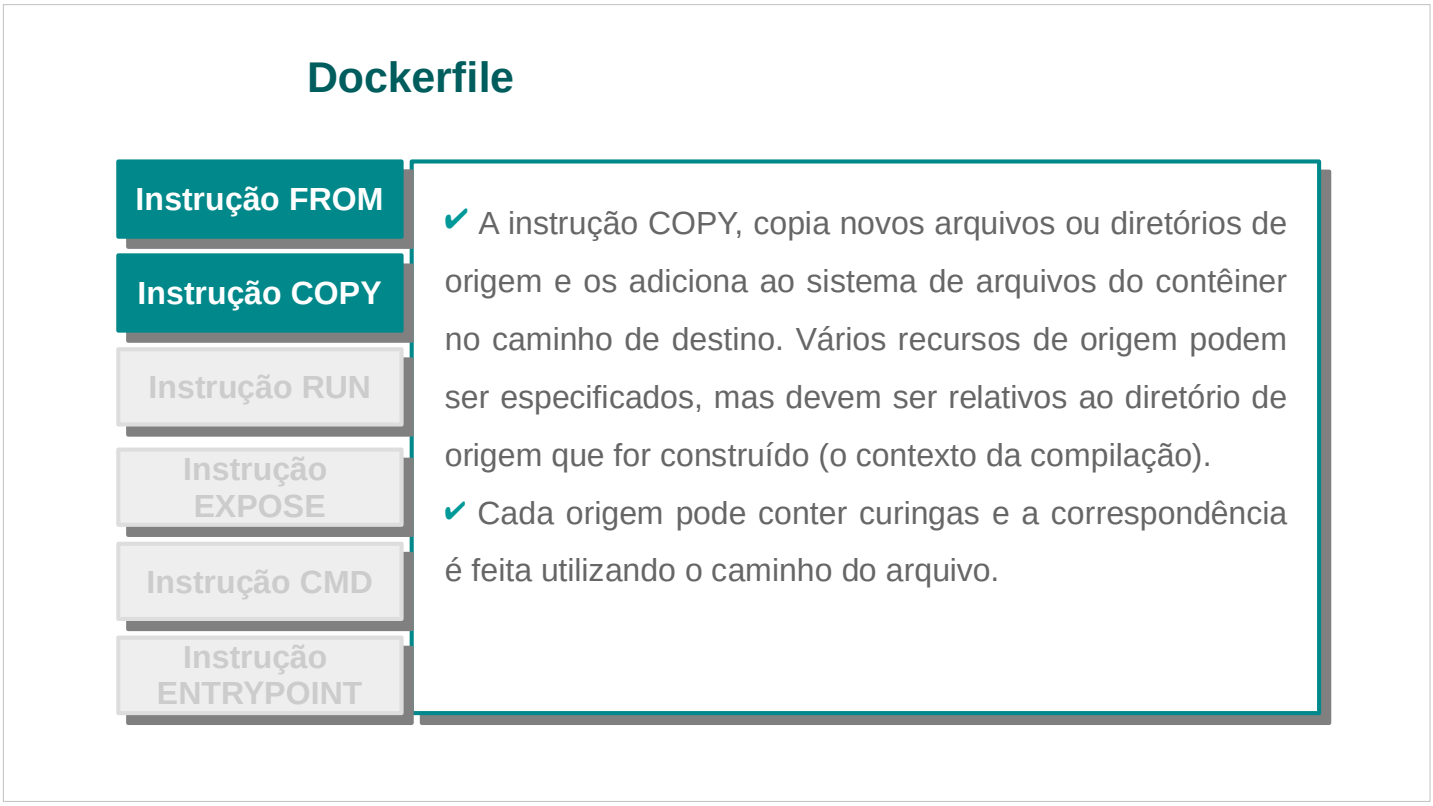
Dockerfile



FROM pode aparecer várias vezes em um único Dockerfile, para criar várias imagens ou usar um estágio de construção como uma dependência para outro. Simplesmente, anote a última saída de ID da imagem pelo commit, antes de cada nova instrução FROM. Cada instrução FROM apaga qualquer estado criado pelas instruções anteriores.

Opcionalmente, um nome pode ser dado a um novo estágio de construção, adicionando o nome AS à instrução FROM. O nome pode ser usado nas instruções FROM e COPY --from = <name | index>, subsequentes para se referir à imagem criada nesta etapa.

Sempre que possível, use Repositórios Oficiais atuais como base para sua imagem. Recomendamos a imagem da Alpine, pois ela é muito controlada e mantida em um nível mínimo (atualmente abaixo de 5 mb), embora continue sendo uma distribuição completa.



Ao copiar arquivos ou diretórios que contenham caracteres especiais, você precisa escapar desses caminhos seguindo as regras do Golang, evitando que eles sejam tratados como um padrão correspondente.

Por exemplo:

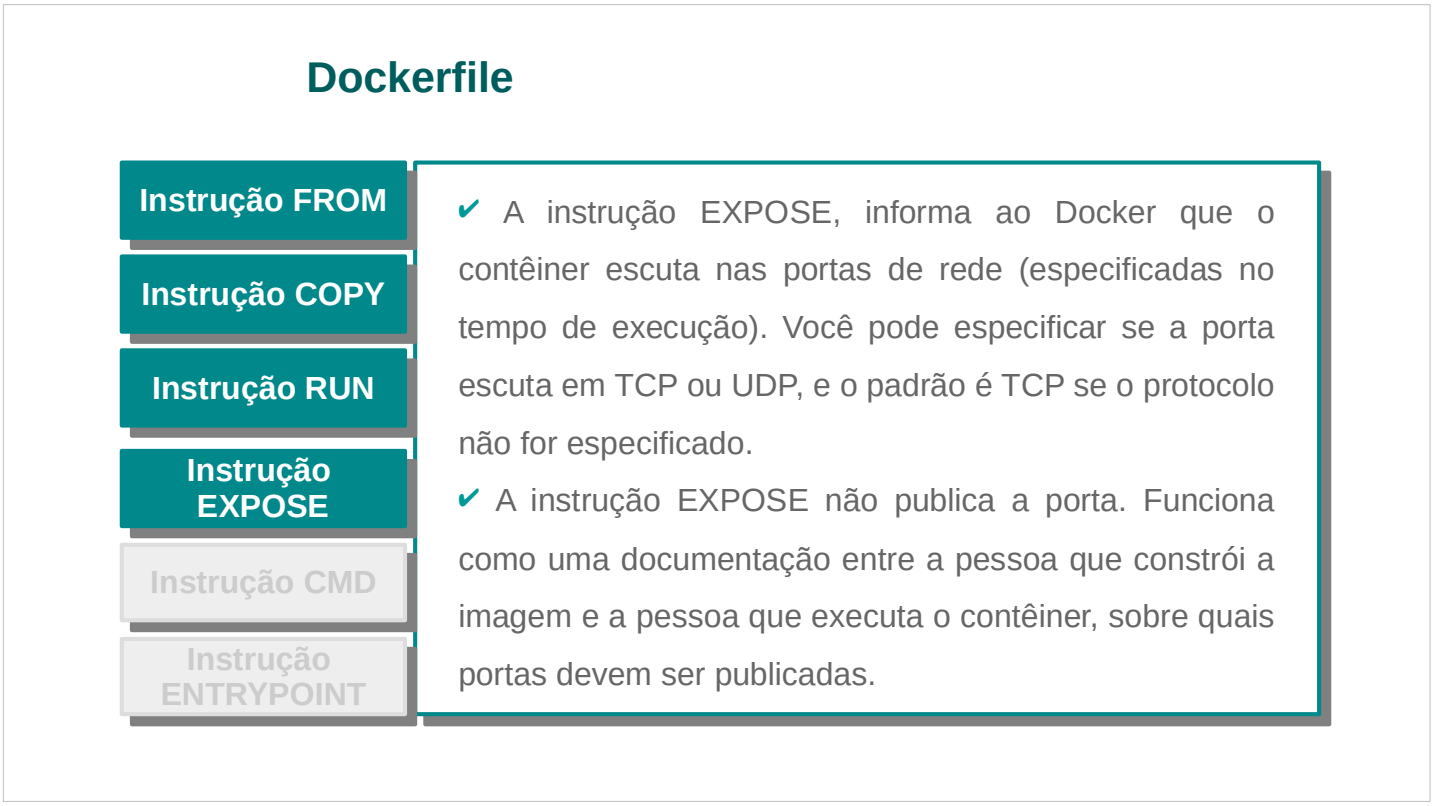
COPY arr[[]0].txt /mydir/

Neste exemplo, seria copiado um arquivo chamado "arr[0].txt" para /mydir.

Sempre combine o RUN apt-get update com o apt-get install na mesma instrução RUN.

Exemplo:

```
RUN apt-get update && apt-get install -y \
    pacote1 \
    pacote2 \
    pacote3
```



A instrução EXPOSE, indica as portas nas quais um contêiner escutará as conexões. Consequentemente, você deve usar a porta comum e a tradicional para seu aplicativo. Por exemplo, uma imagem contendo o servidor web Apache usaria EXPOSE 80, enquanto uma imagem contendo MongoDB usaria EXPOSE 27017 e assim por diante.

Para acesso externo, seus usuários podem executar o docker com um sinalizador, indicando como mapear a porta especificada para a porta de sua escolha. Para vinculação de contêiner, o Docker fornece variáveis de ambiente para o caminho do contêiner do destinatário, de volta para o de origem.

Exemplo:

MYSQL_PORT_3306_TCP

Ao contrário dos comandos em shell, o exec não invoca um shell de comando. Isso significa que o processamento normal do shell não acontece. Por exemplo, CMD ["echo", "$HOME"] não fará substituição de variável em $HOME.

Se você quiser processamento de shell, use o shell ou execute um shell diretamente, por exemplo: CMD ["sh", "-c", "echo $HOME"]. Ao usar o exec e executar um shell diretamente, como no caso do shell, é o shell que fará a expansão da variável de ambiente, não o docker.

A instrução CMD deve ser usada para executar o software contido em sua imagem, junto com quaisquer argumentos. O CMD quase sempre deve ser usado na forma de CMD ["executable", "param1", "param2"…]. Assim, se a imagem for para um serviço, como Apache e Rails, você executaria algo como CMD ["apache2", "- D FOREGROUND"]. De fato, esta forma da instrução é recomendada para qualquer imagem baseada em serviço.

Dockerfile

- Instrução FROM
- Instrução COPY
- Instrução RUN
- Instrução EXPOSE
- Instrução CMD
- Instrução ENTRYPOINT

✔ Os argumentos da linha de comando docker run <image>, serão anexados após todos os elementos em um exec ENTRYPOINT substituírem os elementos especificados, usando o CMD.

✔ Isso permite que os argumentos sejam passados para o ponto de entrada, ou seja, o docker run <image> -d passará o argumento -d para o ponto de entrada. Você pode substituir a instrução ENTRYPOINT, usando a flag -entrypoint no comando docker run.

O melhor uso do ENTRYPOINT é definir o comando principal da imagem, permitindo que essa imagem seja executada como se fosse esse comando (Use o CMD como sinalizadores padrão).

Exemplo:

```
ENTRYPOINT ["s3cmd"]
CMD ["--help"]
```

Criando o Primeiro Dockerfile

1 Conecte via SSH na máquina container e troque para o usuário root:

```
$ vagrant ssh container
$ sudo su -
```

2 Vamos criar um diretório para armazenar o dockerfile:

```
# mkdir -p /root/dockerfiles/echo-container
# cd /root/dockerfiles/echo-container
```

3 Crie um arquivo chamado Dockerfile:

```
# vim Dockerfile
FROM alpine
ENTRYPOINT ["echo"]
CMD ["--help"]
```

container.4labs.example

Anotações

Criando o Primeiro Dockerfile

1 Para criar a imagem a partir do dockerfile, utilize o comando docker image build:

```
# docker image build -t echo-container .
# docker image ls
```

2 Execute um container com a imagem criada:

```
# docker container run --rm -it echo-container
```

3 Execute um container com a imagem criada, alterando o CMD:

```
# docker container run --rm -it echo-container Container DevOps
```

container.4labs.example

Comandos

- ✔ **docker image build**: Permite construir uma imagem do Docker, a partir de um arquivo Dockerfile.

Dockerfile Servidor WEB

1 Vamos criar um diretório para armazenar o dockerfile:

```
# mkdir -p /root/dockerfiles/webserver
# cd /root/dockerfiles/webserver
```

2 Crie um arquivo chamado Dockerfile:

```
# vim Dockerfile
FROM ubuntu
RUN     apt update; \
        apt install wget git apache2 -yq
EXPOSE 80
CMD ["apachectl", "-D", "FOREGROUND"]
```

container.4labs.example

Instruções do Dockerfile

- ✔ **EXPOSE**: Informa ao Docker que o container escuta em uma determinada porta de rede em tempo de execução;
- ✔ **CMD**: Define um comando que será executado no momento da execução do container.

Dockerfile Servidor WEB

3 Para criar a imagem a partir do dockerfile, utilize o comando docker image build:

```
# docker image build -t webserver .
# docker image ls
```

container.4labs.example

Anotações

Recapitulando

1. Dockerfile
2. Sintaxe
3. Criando o Primeiro Dockerfile
4. Dockerfile Servidor WEB

Anotações

Docker Containers

Docker Volumes e Networks

Anotações

Objetivos da aula

1. Docker Volumes
2. Gerenciar Volumes no Docker
3. Docker Networks
4. Administrando Redes

Anotações

Docker Volumes

Definição

✔ Volume é um diretório especialmente designado, seja em um ou mais contêineres que ultrapassam o sistema de arquivos da Union.

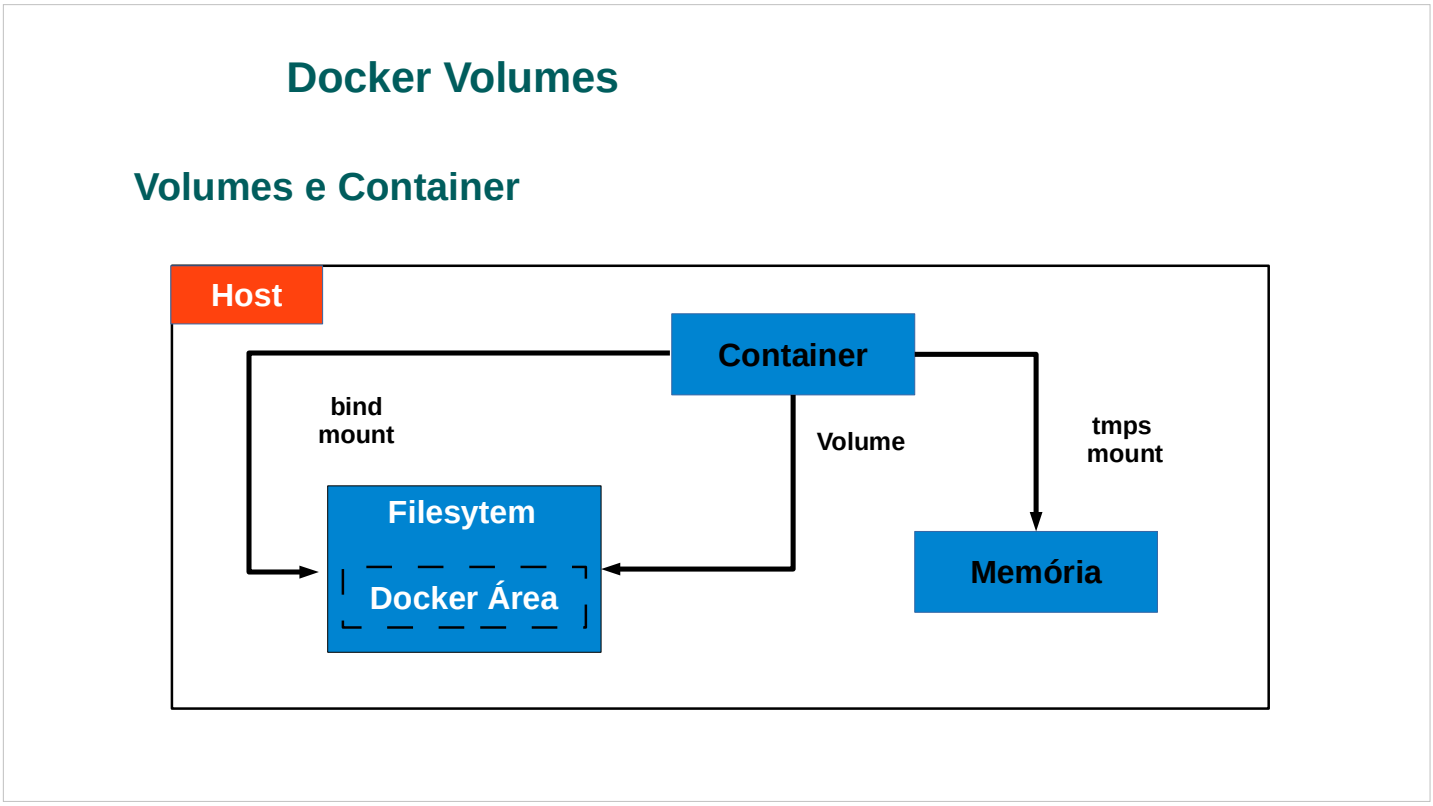
✔ Os volumes são projetados para manter os dados, independentemente do ciclo de vida do contêiner. O Docker, portanto, nunca exclui volumes automaticamente quando você remove um contêiner.

Docker Volumes

Existem três tipos de volumes: host, anônimo e nomeado.

- ✔ **host:** Um volume do host, reside no sistema de arquivos do host do Docker e pode ser acessado de dentro do contêiner;
- ✔ **anônimo:** Um volume nomeado, é um volume no qual o Docker gerencia no disco. O volume é criado, mas recebe um nome;
- ✔ **nomeado:** Um volume anônimo, é semelhante a um volume nomeado, no entanto, pode ser difícil referir-se ao mesmo volume ao longo do tempo, quando se trata de volumes anônimos. Identificador do Docker, onde os arquivos são armazenados.

Os volumes são os mecanismos preferidos para persistir dados gerados e usados pelos contêineres do Docker. Embora as montagens de ligação sejam dependentes da estrutura de diretórios da máquina host, os volumes são completamente gerenciados pelo Docker.



Docker Volumes

Vantagens de um volume sobre montagens bind:

- ✔ Volumes são mais fáceis de fazer backup ou migrar do que montagens de ligação;
- ✔ Você pode gerenciar volumes, usando os comandos do Docker CLI ou a API do Docker;
- ✔ Os volumes funcionam nos contêineres do Linux e do Windows;
- ✔ Os volumes podem ser compartilhados com mais segurança entre vários contêineres;
- ✔ Os drivers de volume, permitem armazenar volumes em hosts remotos ou provedores de nuvem, criptografar o conteúdo de volumes ou adicionar outras funcionalidades;
- ✔ Novos volumes podem ter seu conteúdo pré-preenchido por um contêiner.

Além disso, os volumes costumam ser uma opção melhor do que os dados persistentes na camada gravável de um contêiner, porque um volume não aumenta o tamanho dos contêineres que eles usam, e o conteúdo do volume existe fora do ciclo de vida de um determinado contêiner.

Se o seu contêiner gerar dados de estado não persistentes, considere usar uma montagem tmpfs para evitar o armazenamento permanente dos dados e aumentar o desempenho do contêiner, evitando gravar na camada gravável do contêiner.

Gerenciar Volumes no Docker

Mapear Volumes

1 Para mapear um volume no container com um diretório no host hospedeiro, use a flag “-v” do comando docker run:

```
# docker container run -di --name=servidor -h servidor \
-v /srv:/srv ubuntu
# docker container exec servidor df -Th
```

2 Faça um teste copiando arquivos no host hospedeiro, e verifique se o conteúdo é mapeado no volume do container:

```
# docker container exec servidor ls /srv
# sudo cp /etc/*.conf /srv
# docker container exec servidor ls /srv
```

container.4labs.example

Comandos

- ✔ **docker run -v /volume:/volume:** Permite criar e mapear um novo volume, no momento da execução de um container;
- ✔ **docker container exec:** Permite executar um comando, diretamente em um container.

Gerenciar Volumes no Docker

Criar Volumes com Docker – ETAPA 1

1 Remova a execução de qualquer container servidor para continuar os próximos comandos:

```
# docker container rm -f servidor
```

2 Para criar um volume no container, use a flag “-v” do comando docker run:

```
# docker container run -di --name servidor --hostname servidor \
-v /volume ubuntu
# docker container ls
```

3 Para exibir informações detalhadas sobre o volume, execute o seguinte comando:

```
# docker container inspect servidor | grep volume
```

container.4labs.example

Anotações

Gerenciar Volumes no Docker

Criar Volumes com Docker – ETAPA 2

4	Veja que o Docker criou um hash representando o novo volume criado:
	`# docker volume ls`
	`# docker volume inspect <hash>`

5	Veja também no interior do container o volume criado, através do comando docker exec:
	`# docker container exec servidor df -Th`
	`# docker container exec servidor ls volume`

6	Copie arquivos no host hospedeiro, depois veja se o conteúdo é mapeado no volume:
	`# docker volume ls`
	`# sudo cp /etc/*.conf /var/lib/docker/volumes/<hash>/_data`
	`# docker container exec servidor ls /volume`

container.4labs.example

Descrição dos comandos

- ✔ **docker volume:** Permite gerenciar volumes no Docker;
- ✔ **docker volume ls:** Lista os volumes disponíveis no Docker;
- ✔ **docker volume inspect:** Exibe informações detalhadas sobre um ou mais volumes.

Gerenciar Volumes no Docker

Remover Volumes

1. Remova a execução de qualquer container servidor para continuar os próximos comandos:

```
# docker container rm -f servidor
```

2. Para remover um volume no Docker, usamos a opção “rm”, conforme o exemplo abaixo:

```
# docker volume ls
# docker volume rm <hash>
# docker volume ls
```

container.4labs.example

Descrição dos comandos

- ✔ **docker volume rm:** Permite remover um volume disponível no Docker.

Docker Network

Definição

✔ O Docker utiliza recursos de redes para fornecer isolamento completo aos containers, então, eles se conectam à rede e de lá para onde vão?

✔ Bem, eles podem ir para a internet. Talvez haja aplicativos de entrada que precisam de acesso ao aplicativo em execução no contêiner, ou talvez o aplicativo precise de acesso a algo na internet para fazer algum tipo de processamento e apresentar alguma informação.

container.4labs.example

Docker Network

A filosofia de rede do Docker é baseada em aplicativos. O objetivo é fornecer opções e flexibilidade para os operadores de rede, bem como o nível correto de abstração para os desenvolvedores de aplicativos.

Como qualquer design, o design de rede é um ato de equilíbrio. O Docker EE e o ecossistema do Docker, fornecem várias ferramentas aos engenheiros de rede para obter o melhor equilíbrio para seus aplicativos e ambientes. Cada opção oferece benefícios e compensações diferentes.

Recursos de rede do Docker EE:

Os dois recursos a seguir, só são possíveis ao utilizar o Docker EE e gerenciar seus serviços do Docker usando o UCP:

- ✔ A malha de roteamento HTTP, permite que você compartilhe o mesmo endereço IP de rede e porta, entre vários serviços. O UCP encaminha o tráfego para o serviço apropriado, usando a combinação de hostname e port, conforme solicitado pelo cliente;
- ✔ A rigidez de sessão, permite especificar informações no cabeçalho HTTP que o UCP utiliza para rotear solicitações subsequentes para a mesma tarefa de serviço, e para aplicativos que exigem sessões com preservação de estado.

Docker Network

Drivers de rede nativa do Docker

✔ Os drivers de rede nativos do Docker, fazem parte do Docker Engine e não exigem módulos extras. Eles são chamados e usados por meio de comandos padrão da rede do docker.

✔ Os seguintes drivers de rede nativos estão disponíveis no Docker: Host, Bridge, Overlay, MACVLAN ou Nenhum(None).

✔ Neste curso trabalharemos com as redes Host e Bridge.

container.4labs.example

Descrição do drivers de rede

- ✔ **Host:** Com o driver do host, um contêiner usa a pilha de rede do host. Não há separação de namespace e todas as interfaces no host podem ser usadas diretamente pelo contêiner;
- ✔ **Bridge:** O driver de ponte, cria uma ponte do Linux no host que é gerenciado pelo Docker. Por padrão, contêineres em uma ponte podem se comunicar uns com os outros. O acesso externo a contêineres, também pode ser configurado através do driver da ponte;
- ✔ **Overlay:** O driver de sobreposição, cria uma rede de sobreposição que suporta redes de vários hosts prontas para uso. Ele usa uma combinação de pontes Linux locais e VXLAN para sobrepor comunicações de contêiner à contêiner, sobre infra-estrutura de rede física;
- ✔ **MACVLAN:** O driver macvlan, usa o modo de ponte MACVLAN para estabelecer uma conexão entre interfaces de contêiner e uma interface de host pai (ou subinterfaces). Ele pode ser usado para fornecer endereços IP para contêineres que são roteáveis na rede física;
- ✔ **Nenhum:** O driver none, fornece ao contêiner seu próprio stack de rede e namespace de rede, mas não configura interfaces dentro do contêiner. Sem configuração adicional, o contêiner é completamente isolado da pilha de rede do host;
- ✔ **Plugins de rede:** Você pode instalar e usar plug-ins de rede de terceiros com o Docker. Esses plug-ins estão disponíveis no Docker Hub ou em fornecedores de terceiros.

Introdução à Redes no Docker

Rede Host

container.4labs.example

Rede Host

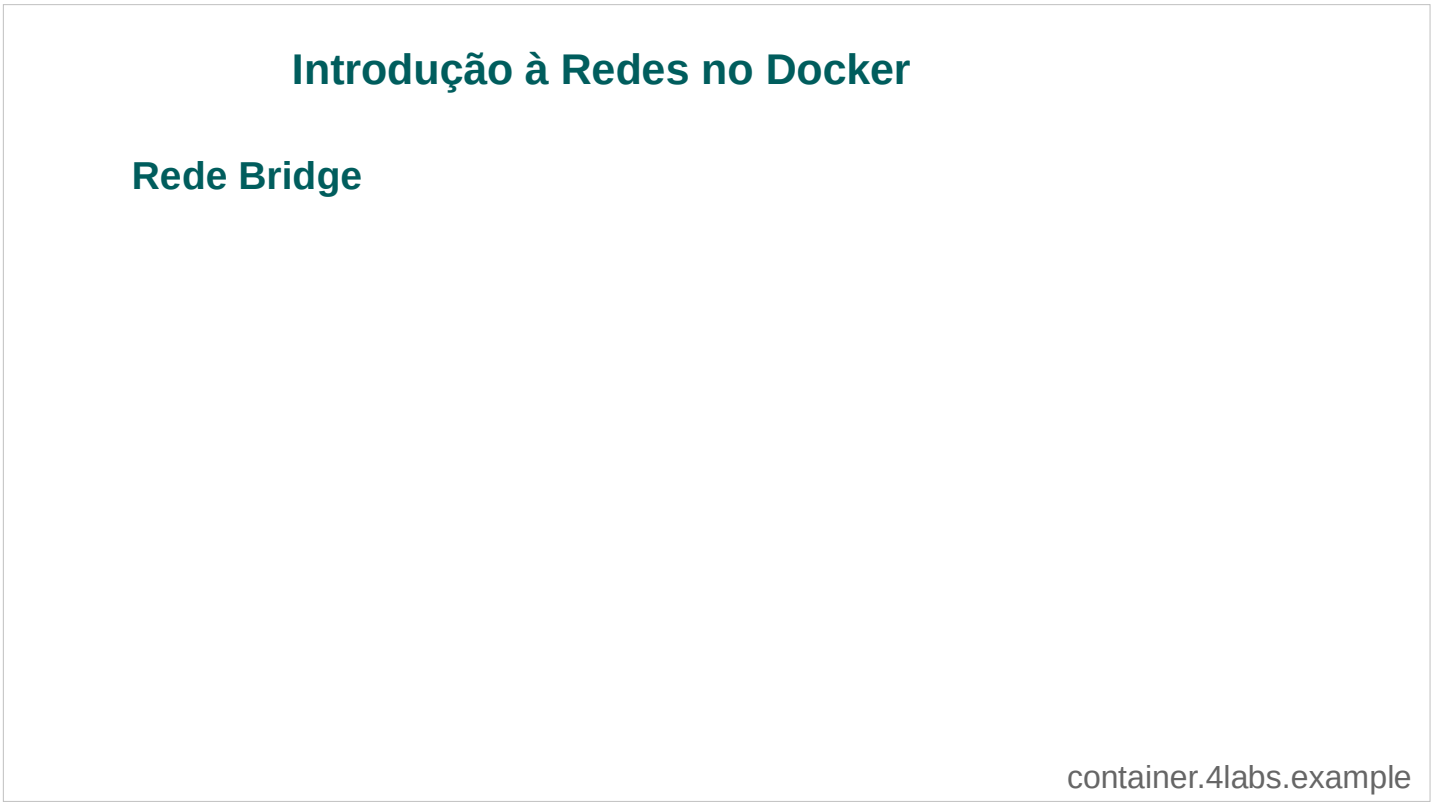
O driver de rede do host, é a mesma configuração de rede que o Linux usa sem o Docker. A flag –net=host efetivamente desativa a rede do Docker, e os contêineres usam a pilha de rede do host (ou padrão) do sistema operacional do host.

Normalmente, com outros drivers de rede, cada contêiner é colocado em seu próprio namespace de rede (ou sandbox) para fornecer isolamento de rede completo entre si. Com os contêineres do driver host, todos estão no mesmo namespace de rede do host e usam as interfaces de rede e a pilha IP do host.

Todos os contêineres na rede do host podem se comunicar entre si nas interfaces do host. Do ponto de vista da rede, isso equivale a vários processos em execução de um host sem contêineres. Como eles, estão usando as mesmas interfaces de host, não há dois contêineres capazes de se vincular à mesma porta TCP. Isso pode causar contenção de porta, se vários contêineres estiverem sendo planejados no mesmo host.

Introdução à Redes no Docker

Rede Bridge



Rede Bridge

Em qualquer host que esteja executando o Docker Engine, existe por padrão, uma rede Docker local chamada bridge. Essa rede é criada usando um driver de rede de ponte, que instancia uma ponte do Linux chamada docker0.

Em um host independente do Docker, bridge é a rede padrão à qual os contêineres se conectam, se nenhuma outra rede for especificada. O Docker Engine conecta-o à rede de bridge por padrão.

Descrição

- ✔ **bridge:** É o nome da rede Docker;
- ✔ **bridge:** É o driver de rede ou modelo, a partir do qual esta rede é criada;
- ✔ **docker0:** É o nome da ponte Linux que é o bloco de construção do kernel, usado para implementar esta rede.

Portas

✔ Por padrão, quando você cria um contêiner, ele não publica nenhuma de suas portas para o mundo externo. Para disponibilizar uma porta para serviços fora do Docker ou para contêineres do Docker que não estejam conectados à rede do contêiner, use a flag --publish ou -p. Isso cria uma regra de firewall que mapeia uma porta de contêiner, para uma porta no host do Docker.

container.4labs.example

Anotações

Administrando Redes

Gerenciar mapeamento de portas

1 Para mapear a porta 80 do container no host local, execute um container com a “flag -p”:

```
# docker container run -d --name webserver -p 80:80 webserver
```

2 Para verificar qual é a porta que está sendo mapeada no container, execute o comando:

```
# docker container port webserver
```

Faça um teste acessando o servidor Web Apache, através da máquina local:

```
http://container.4labs.example
```

3 Remova a execução do container “webserver”, antes de continuar os próximos comandos:

```
# docker container rm -f webserver
```

container.4labs.example

Anotações

Quando executamos um container, podemos definir a exposição de uma ou mais portas, através da opção -p host:port:port para especificar a interface externa para uma determinada ligação.

Para expor as portas em qualquer endereço, use o endereço 0.0.0.0 no host hospedeiro. Através do comando docker container port, podemos exibir a porta mapeada no container.

Gerenciar Redes Bridge no Docker

Executar container com rede bridge

1. Para utilizar a rede bridge, execute o container com a flag --network bridge:

```
# docker container run -d --name webserver --network bridge \
-p 80:80 webserver
```

2. Verifique qual é a porta que está sendo mapeada no host local:

```
# docker container port webserver
# sudo ss -ntpl | grep 80
```

3. Faça um teste acessando o Apache do navegador, depois remova a execução do mesmo:

```
http://container.4labs.example
# docker container rm -f webserver
```

container.4labs.example

Anotações

Quando utilizamos a flag -d bridge, o container roda com a rede padrão que o Docker configura automaticamente para você. Esta rede não é a melhor escolha para sistemas de produção.

Gerenciar Redes Bridge no Docker

Executar container com rede host

1	Para executar um container utilizando a rede host, execute o container com a flag --net: `# docker container run -d --name webserver --net host webserver`

2	Veja se o container não possui porta mapeada, pois o ”Apache” está utilizando a porta diretamente no host local:
	`# docker container port webserver`
	`# sudo ss -ntpl | grep 80`

3	Faça um teste acessando o servidor Web Apache, através do Navegador: `http://container.4labs.example`

container.4labs.example

Anotações

Quando utilizamos a flag –net=host, estamos iniciando um contêiner que se liga diretamente à porta no host do Docker. Do ponto de vista da rede, esse é o mesmo nível de isolamento como se o processo nginx estivesse sendo executado diretamente no host do Docker e não em um contêiner.

No entanto, em todas as outras formas, como armazenamento, namespace de processo e namespace de usuário, o processo nginx é isolado do host.

Gerenciar Redes Bridge no Docker

1 Vamos testar o uso da rede padrão com dois containers, através da imagem “Debian”:

```
# docker container run -di --name c1 -h servidor debian
# docker container run -di --name c2 -h cliente debian
```

2 Verifique o endereço IP de cada container, através do comando “docker container exec”:

```
# docker container exec c1 ip -c a show eth0
# docker container exec c2 ip -c a show eth0
```

3 Faça um teste de conectividade entre os containers, através do comando “ping”:

```
# docker container exec c1 ping -c4 172.17.0.3
# docker container exec c2 ping -c4 172.17.0.2
```

Worker 01

container.4labs.example

Anotações

Gerenciar Redes Bridge no Docker

Criar Redes Personalizadas – ETAPA 1

1 Remova a execução dos containers c1 e c2, antes de continuar os próximos comandos:

```
# docker container rm -f c1 c2
```

2 Como exemplo prático, vamos criar uma nova rede no Docker com o nome de “4labslan”:

```
# docker network create --driver bridge --subnet 172.19.0.0/16 \
4labslan
# docker network ls
```

3 Para exibir informações detalhadas sobre a rede “4labslan”, execute o seguinte comando:

```
# docker network inspect 4labslan
```

container.4labs.example

Descrição dos comandos

- ✔ **docker network create**: Permite criar uma nova rede no Docker. A opção –subnet define a rede e máscara.

Gerenciar Redes Bridge no Docker

Criar Redes Personalizadas – ETAPA 2

4 Inicie o primeiro container de nome c1 na rede “4labslan”, através da opção --net:

```
# docker container run -di --name=c1 -h servidor --net 4labslan \
debian
```

5 Inicie o segundo container de nome c2 na rede “4labslan”, através da opção --net:

```
# docker container run -di --name=c2 -h cliente --net 4labslan \
debian
```

6 Faça um teste de conectividade, usando o hostname de cada container na rede “dexterlan”:

```
# docker container exec c1 ping -c4 c2
# docker container exec c2 ping -c4 c1
```

container.4labs.example

Anotações

Gerenciar Redes Bridge no Docker

Utilizar IP Fixo – ETAPA 1

1 Remova a execução dos containers c1 e c2, antes de continuar os próximos comandos:

```
# docker container rm -f c1
# docker container rm -f c2
```

2 Inicie novamente os dois containers na rede “4labslan”, utilizando IP fixo:

```
# docker container run -di --name=c1 -h servidor --net 4labslan \
--ip 172.19.0.111 debian
# docker container run -di --name=c2 -h cliente --net 4labslan \
--ip 172.19.0.112 debian
```

container.4labs.example

Descrição dos comandos

- ✔ A opção –ip só pode ser utilizada com a opção –net, em uma rede definida pelo usuário.

Gerenciar Redes Bridge no Docker

Utilizar IP Fixo – ETAPA 2

3	Faça um teste de conectividade, usando o hostname de cada container na rede “4labslan”:
	`# docker container exec c1 ping -c4 172.19.0.112`
	`# docker container exec c2 ping -c4 172.19.0.111`

4	Para desconectar um container de uma rede, use a opção “disconnect”:
	`# docker network disconnect 4labslan c2`
	`# docker container exec c1 ping -c4 172.19.0.112`

5	Para conectar um container a uma rede, use a opção “connect”:
	`# docker network connect --ip 172.19.0.112 4labslan c2`
	`# docker container exec c1 ping -c4 172.19.0.112`

container.4labs.example

Descrição dos comandos

- ✔ **docker network disconnect:** Permite desconectar um container de uma rede, disponível no Docker;
- ✔ **docker network connect:** Permite conectar um container a uma rede, disponível no Docker.

Gerenciar Redes Bridge no Docker

Remover Rede

1 Remova a execução dos containers c1 e c2, antes de continuar os próximos comandos:

```
# docker container rm -f c1
# docker container rm -f c2
```

2 Para remover uma rede no Docker, usamos a opção “rm”, conforme o exemplo abaixo:

```
# docker network rm 4labslan
# docker network ls
```

container.4labs.example

Descrição dos comandos

- ✔ **docker network rm:** Permite remover uma rede, disponível no Docker.

Recapitulando

1. Docker Volumes
2. Gerenciar Volumes no Docker
3. Docker Networks
4. Administrando Redes

Anotações

Docker Containers

Docker Compose

Anotações

Objetivos da aula

1. Docker Compose
2. Etapas de um Compose
3. docker-compose.yml
4. Criando Composes

Anotações

Docker Compose

Compose é uma ferramenta para definir e executar aplicações multi-contêineres em Docker. Com um simples comando, você consegue criar e iniciar todos os serviços através de sua configuração.

Ela funciona em todos os ambientes: produção, homologação, desenvolvimento, teste, bem como em fluxos de Integração Contínua.

Anotações

Etapas de um Compose

Utilizar o Compose é basicamente um processo de três etapas:

1. Definir seu ambiente com um Dockerfile para que ele possa ser reproduzido em qualquer lugar;
2. Definir os serviços que serão executados em um arquivo docker-compose.yml;
3. Executar o comando docker-compose up.

Anotações

docker-compose.yml

- O arquivo **docker-compose.yml** define a estrutura e como serão executados os contêineres;
- É importante notar que as versões do arquivo **docker-compose** podem não ser compatíveis com determinados parâmetros, e até mesmo versões diferentes da Docker Engine.
- Podemos visualizar as referências do arquivo docker.compose.yml, diretamente na documentação do docker na seção **Compose file reference**.
- https://docs.docker.com/compose/compose-file/

Anotações

docker-compose.yml

O docker-compose.yml é apresentado da seguinte maneira:

Obrigatoriamente, o docker-compose.yml deve começar indicando qual a versão do compose a ser utilizada. Em seguida, podemos definir os serviços, volumes e redes não importando a ordem. Cada container a ser executado é uma nova seção do grupo **services.**

Criando Composes

1 Primeiramente, conecte-se ao servidor container como root:

```
$ vagrant ssh container
$ sudo su -
```

2 Antes de executar qualquer comando, vamos remover todos os containers existentes:

```
# docker container rm -f $(docker container ls -aq)
```

3 Vamos criar um diretório para armazenar o arquivo docker-compose:

```
# mkdir /root/compose
# cd /root/compose
```

4 Copie o dockerfile do nosso webserver para a pasta do compose:

```
# cp /root/dockerfiles/webserver/Dockerfile .
```

container.4labs.example

Descrição dos comandos

- ✔ **docker container rm -f:** Força a remoção de todos os containers;
- ✔ **docker container ls -aq:** Lista o id de todos os containers da máquina.

Criando Composes

1 Crie o arquivo docker-compose.yml:

```
# vim docker-compose.yml

version: '3'
services:
  webserver:
    build: .
    hostname: webserver
    ports:
      - 80:80
    restart: always
```

2 Execute o docker compose com o comando:

```
# docker-compose up -d
```

container.4labs.example

IMPORTANTE: Por padrão, o comando docker-compose procura por um arquivo docker-compose.yml no diretório em que o comando foi executado. É possível informar qual é o arquivo compose a ser executado, através da opção **-f <caminho-do-compose>**

Parâmetros do docker-compose.yml

version: Define a versão do arquivo compose;
services: Define a seção de serviços;
build: Define o local do Dockerfile;
hostname: Define o hostname do container criado;
ports: Define quais portas serão publicadas;
restart: Define a polÍtica de restart.

Comandos

✔ **docker-compose up -d**: Executa o conteúdo do compose de forma detached;

==== docker-compose.yml =====

```
version: '3'
services:
  webserver:
    build: .
    hostname: webserver
    ports:
      - 80:80
    restart: always
```

Criando Composes

Podemos abrir pelo navegador o endereço **http://container.4labs.example** e verificar se o nosso webserver está sendo executado:

1. Para parar o container criado com o compose, utilize o comando:

```
# docker-compose stop
# docker container ls -a
```

2. Para iniciar novamente um container criado com o compose, utilize o comando:

```
# docker-compose start
# docker container ls -a
```

3. Para destruir um container criado com o compose, utilize o comando:

```
# docker-compose down
# docker container ls -a
```

container.4labs.example

Comandos

- ✔ **docker-compose stop:** Pára um compose que está em execução;
- ✔ **docker-compose start:** Inicia um compose parado;
- ✔ **docker-compose down:** Destrói o ambiente criado, através do compose.

Criando Composes

Vamos agora criar um arquivo **html** e definir um volume para o nosso container, através do **docker-compose**.

1 Crie a pasta html e o arquivo index.html:

```
# mkdir html
# echo "<h1> Website 4Labs </h1>" > html/index.html
```

2 Edite o compose e adicione as seguintes linhas no final do arquivo, respeitando a identação:

```
# vim docker-compose.yml
(...)
    restart: always
    volumes:
      - $PWD/html:/var/www/html
```

container.4labs.example

O parâmetro **$PWD** referencia o **P**arent **W**orking **D**irectory (Diretório atual).

Através deste mapeamento de volumes, estamos substituindo o conteúdo do caminho /var/www/html por /root/compose/html

==== docker-compose.yml =====

```
version: '3'
services:
  webserver:
    build: .
    hostname: webserver
    ports:
      - 80:80
    restart: always
    volumes:
      - $PWD/html:/var/www/html
```

Criando Composes

Execute o compose e acesse o website http://container.4labs.example.

1 Execute o compose:

```
# docker-compose up -d
```

container.4labs.example

Anotações

Criando Composes

1 Destrua o compose criado:

```
# docker-compose down
```

Todos os parâmetros utilizados no comando **docker container run** podem ser reescritos em um compose, tornando assim mais fácil a criação e gestão de conteiners.

container.4labs.example

O parâmetro **$PWD** referencia o **P**arent **W**orking **D**irectory (Diretório atual).

Através deste mapeamento de volumes, estamos substituindo o conteúdo do caminho /var/www/html por /root/compose/html

Criando Composes

- ✔ A funcionalidade principal do docker compose, é subir serviços multi-containers, como por exemplo um serviço de wordpress, onde será executado um container para aplicação e um segundo container para o banco de dados.
- ✔ Fazemos isto, definindo múltiplos segmentos dentro de um docker-compose.

Anotações

Criando Composes

Gere um novo compose para o nosso serviço wordpress + mysql utilizando o modelo da pasta Aula 9.5:

```
# cp /vagrant/4525/arquivos/Aula\ 9.5/14_wordpress-compose.yml wordpress-compose.yml
# cat wordpress-compose.yml
version: '3'
volumes:
  mysql_db:
services:
  wordpress:
    image: wordpress
    restart: always
    ports:
      - 80:80
    environment:
 (...)
```

container.4labs.example

Parâmetros do wordpress-compose.yml

image: Define a imagem a ser utilizada no serviço;
environment: Define as variáveis de ambiente do serviço;
volumes (seção): Define a criação de um volume.

===== wordpress-compose.yml ======

```
version: '3'
volumes:
  mysql_db:
services:
  wordpress:
    image: wordpress
    restart: always
    ports:
      - 80:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wpuser
      WORDPRESS_DB_PASSWORD: devops@4linux
      WORDPRESS_DB_NAME: wordpress
  db:
    image: mysql:5.7
    restart: always
    volumes:
      - mysql_db:/var/lib/mysql
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wpuser
      MYSQL_PASSWORD: devops@4linux
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
```

Criando Composes

1 Execute o compose:

```
# docker-compose -f wordpress-compose.yml up -d
```

2 Verifique os containers criados:

```
# docker container ls
```

container.4labs.example

Parâmetros

-f <arquivo>: Especifica o arquivo **.yml** para ser utilizado como compose.

Criando Composes

Acesse o endereço **http://container.4labs.example** e configure o wordpress:

Anotações

Criando Composes

Anotações

Criando Composes

Anotações

Criando Composes

Vamos acessar agora o wordpress
e efetuar algumas modificações:

Anotações

Criando Composes

Vamos alterar o tema do Wordpress, para isto, clique em **Aparência** e depois em **Temas**:

Anotações

Criando Composes

Selecione um tema e clique em **Ativar**:

Anotações

Criando Composes

Para visualizar o tema, clique em **Wordpress – 4Labs**:

Anotações

Criando Composes

Verifique se o tema foi alterado:

Anotações

Criando Composes

1	Verifique os logs do container, através do comando docker-compose logs: `# docker-compose -f wordpress-compose.yml logs`
2	É possível visualizar os logs em tempo real, através do comando docker-compose logs -f: `# docker-compose -f wordpress-compose.yml logs -f` <CTRL> + <C>

container.4labs.example

Comandos

- ✔ **docker-compose logs:** Comando equivalente ao cat <arquivo_de_log>;
- ✔ **docker-compose logs -f:** Comando equivalente ao tail -f <arquivo_de_log>.

Criando Composes

1. Destrua o compose:

```
# docker-compose -f wordpress-compose.yml down
```

2. Verifique se o volume ainda existe:

```
# docker volume ls
```

container.4labs.example

Note que ao destruir o compose, os volumes continuam persistentes no disco. Sendo assim, é possível recriá-los com o mesmo conteúdo.

Criando Composes

Tente acessar o website e verifique se o mesmo está indisponível.

Anotações

Criando Composes

1. Execute o compose novamente:

```
# docker-compose -f wordpress-compose.yml up -d
```

Veja se os dados modificados persistiram no endereço http://container.4labs.example.

container.4labs.example

Anotações

Criando Composes

1. Destrua o compose:

```
# docker-compose -f wordpress-compose.yml down
```

2. Remova o volume criado:

```
# docker volume rm compose_mysql_db
```

container.4labs.example

Anotações

Recapitulando

1. Docker Compose
2. Etapas de um Compose
3. docker-compose.yml
4. Criando Composes

Anotações