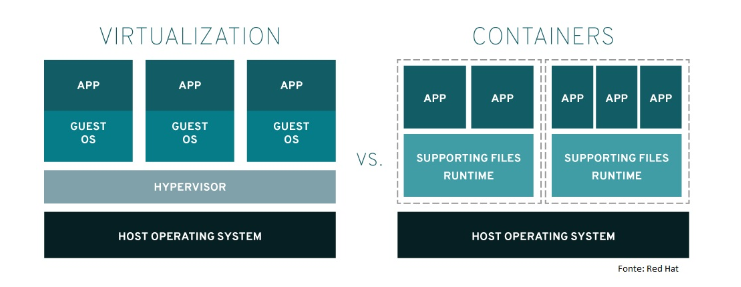
**Docker**

Facilita a criação e empacotamento de uma aplicação dentro de um container, é portátil, qualquer outro host que tenha Docker pode subir sua aplicação, a ideia do Docker é subir apenas uma máquina, ao invés de várias e nessa máquina pode rodar várias aplicações sem que haja conflito entre elas.

Docker não é uma máquina virtual, utiliza containers, que na verdade se trata de um ambiente isolado, o container possui um conjunto de processos que a partir de uma imagem pode ser executado, esta imagem possui todos arquivos necessários para rodar a aplicação, os containers compartilham o mesmo kernel, entretanto os processos são isolados através do namespace.

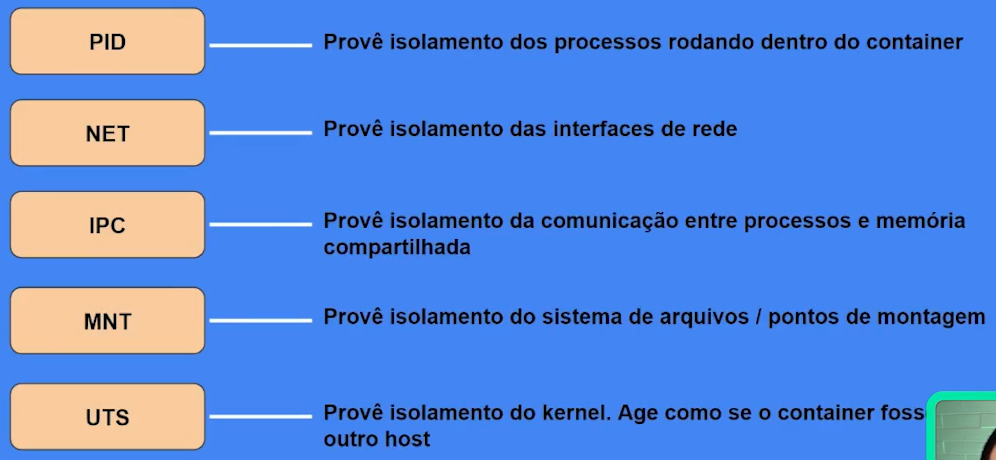


Podemos ver que, na virtualização, temos um maior consumo de recursos, uma vez que para cada aplicação precisamos carregar um sistema operacional. Já no Docker, podemos ver que não existe essa necessidade de múltiplos sistemas operacionais convidados.

Como o container contém todas as dependências de um aplicativo, ele é portátil e consiste em todas as etapas do desenvolvimento, o Docker nos permite construir nossas próprias imagens e utiliza-las como base para os containers.

Vantagens como: economia de recursos, melhor disponibilidade do sistema, possibilidade de compartilhamento e simplicidade de criação.

**Namespace e níveis de isolamento.**

****

Como funciona o processo de virtualização com container -> Se os containers estão rodando em máquina que o kernel do Linux, por exemplo, cada um desses containers carrega um pedaço do kernel também, só que isolado.

**Cgroups**

Gerenciamento de recursos, consegue definir tanto de maneira automática como de maneira manual como que os consumos serão definidos.

ao utilizar o Docker precisamos conhecer e saber para que servem alguns comandos para fazer ações quando trabalhamos com containers, sendo um deles o “**docker run”.**

**O comando Docker run é responsável por executar um container em nosso host.**

Docker HUB -> Um grande repositório de imagens oficiais.

Um container para que esteja em exeução, deve ter no mínimo um processo dentro dele para que ele fique vivo, se não houver processo em execução, ele não vai ficar em execução.

**O que o Docker faz embaixo dos panos quando mandamos um comando**

Se não tem a imagem localmente, o host vai lá no Docker Hub, solicita a imagem, o DC envia a imagem, o host faz uma verificação para certificar que era a imagem que estávamos procurando e depois executa.

**Comandos importantes Docker**

**CTRL + D ->** sai do container.

**Docker stop (container ID)** -> Com esse comando vai parar o comando realizado. Ele “mata um processo”.

**Docker pause (container ID)** -> Com esse comando vai PAUSAR o comando realizado. Ele NÃO“mata um processo”. **Unpause** -> saído pause.

**Docker start (container ID)** -> Ele vai voltar a executar o comando.

**Docker exec -it (container ID) bash** -> -it quer dizer um commando intereativo bash vc vai entrar dentro do terminal do container

**Docker rm (container ID)** -> Remove o container e ele deixa de existir e caso não configure nada pode perder os arquivos.

O docker run cria um novo container e o executa. O docker exec permite executar um comando em um container que já está em execução.

**Para passar um comando que não va travar seu terminal tem que passar a flag “-d” (detect).**

docker rm **container ID** –force -> commando que vai forçar a remoção da aplicação.

-P (maiúsculo) -> Mapeamento de portas meio maluco.

-p(minúsculo) Vc que faz o mapeamento de portas. EX: docker run -d -p 8080:80 dockersamples/static-site (ali ta dizendo que a porta 8080 vai refletir a porta 80 do Docker).

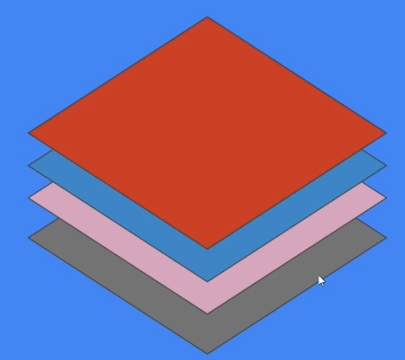
 Parar vários container ao mesmo tempo (-q) significa que ta parando eles por seu id.

docker port **container ID -> mostra o mapeamento de porta. O Docker precisa fazer o mapeamento interno entre a nossa máquina e o container**

**Imagens: o grande pulo do gato do container**

Imagens a grosso modo, pode ser considerado como uma receita para criar um container.

Imagens são formadas por um conjunto de camadas.



As camadas são independentes e cada uma tem seu id.

**docker images ->** Consegue ver quais imagens tem baixado.

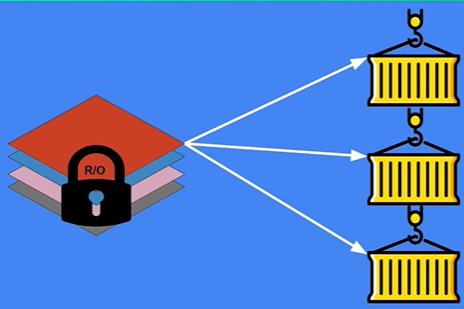
Docker inspect “idimagem” -> Inspeciona a imagem

docker history “idimagem” -> Inspeciona as camadas

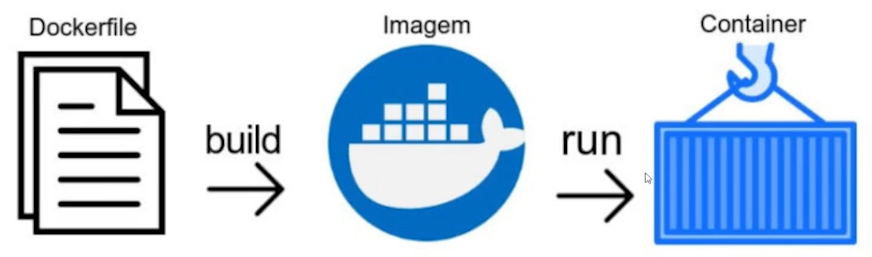
Uma características do Docker é que ele é inteligente suficiente para reutilizar camadas de novas das imagens que já tem, assim melhora a performance, não gera camadas duplicadas. Quando um container é criado, nasce um conjunto de camadas de imagens imutáveis e 1 camada de leitura e escrita.

**Por que o Container é tão leve?**

A imagem é apenas de leitura, nós podemos ter um, dois, 100 ou 10 mil containers baseados na mesma imagem. A diferença é que cada um desses containers vai ter só uma camada diferente de um para o outro de *read-write*.

E como essa camada é extremamente leve, a fim de manter essa performance, nós temos uma reutilização da imagem para múltiplos containers.

**Criando uma imagem**

<-Fluxo para criação de uma imagem

Função do Docker: Ele é um servidor, que disponibiliza a aplicação, cria uma imagem com todas as dependências e arquivos.

No exemplo que estamos criando, o node vai ser responsável por executar nosso servidor e também precisa de um comando que mantenha esse servidor em execução.

Como o professor pensou para construir uma imagem:

1°Precisa de um node para rodar um servidor: Como fazer isso?

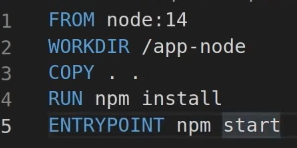
Grande vantagem do Docker, não precisa de um SO para fazer isso e consegue buscar uma imagem oficial do Docker para o projeto, para isso dentro do “DOCKERFILE” vai coloca o comando: FROM node:14(14 é a versão do node).

2°O que queremos fazer: Colocar todos os arquivos da aplicação dentro da imagem, ou seja, COPIAR o conteúdo do Host para a imagem. -> COPY . /app-node(O . significa da pasta atual), /app-node (vai ser o nome da imagem).

WORKDIR ->Define a pasta de trabalho

RUN -> o comando npm install vai ser executado enquanto a imagem esta sendo criada.

E quando o container for executado a partir dessa imagem o comando executado vai ser o npm start



**Comandos que precisam ser dados para criar uma imagem de fato.**

Através do terminal vc precisar estar no diretório que esta seu arquivo Dockerfile

IMPORTANTE: O “.” NO FINAL DO COMANDO INDICA QUE TODO ESSE COMANDO É DADO A PARTIR DA PASTA ATUAL.

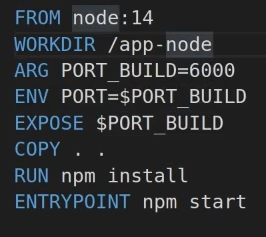
 QUAIS IMAGES TEM NA SUA MÁQUINA



Usando a imagem criada.

 -> Vai expor que a nossa aplicação esta na porta 3000.

Realizar esse procedimento de forma mais parametrizada:

 ARG -> Vai ser parametrizada somente no momento de buil.

ENV-> Se a gente quer manda essa info pro nosso container e fica exposto para quem quiser ver.  
ARG – serve para construção da imagem

ENV – posteriormente dentro do container.

Subindo um container para o Docker Hub.

Precisa fazer o login via prompt 

Para pode enviar tive que mudar o nome do container com o nome de usuário que tenho no Docker Hub



Enviando uma imagem para o Docker Hub: abaixo



 Remover container

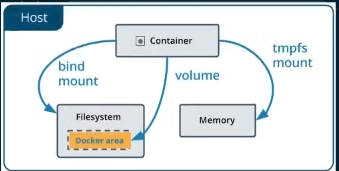
 Remover imagens

Quando containers são removidos, nossos dados são perdidos, Podemos persistir dados em definitivo através de volumes e bind mounts

Bind mounts dependem da estrutura de pastas do host

Volumes são gerenciados pelo Docker

Tmpfs armazenam dados em memória volátil.

Como faz para persistir informações de container que não estão mais em execução?

Utilizando os volumes, com volumes, é possível escrever os dados em uma camada persistente. Com bind mounts, é possível escrever os dados em uma camada persistente baseado na estrutura de pastas do host.

Bind mounts



Com esse comando ele ta rodando um ubuntu e a flag -v indica que vai ter um diretório dentro na pasta tal que os dados serão persistidos



O comando acima é um pouco mais semântico.

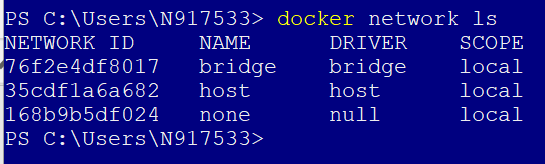
**Utilizando Volumes**

É a maneira mais recomendada, pois quem gerencia seu filesystem é próprio Docker.

**Comunicação através de redes**

Os containers estão na mesma rede, será que conseguimos realizar uma comunicação entre eles?

Com esse comando fez um mapeamento das redes que tem no container



Quando sobe um Docker ele vem configurado por padrão na rede “bridge”.

No exemplo que o professor usou, ele tava com 3 imagens do ubuntu que supostamente que estavam na mesma rede e deveriam se comunicar através do comando ping, só que as imagens não tinham essa imagem (ping) então ele foi instalando via terminal (do container) como se tivesse mexendo em uma máquina Linux.

Comando realizados:

apt-get update -> Atualiza os pacotes

apt-get install iputils-ping -y.

Uma comunicação via ip não é confiável, pois os containers são suscetíveis a reiniciar, trocar o Ip, sendo assim ia ter uma comunicação instável.

Comunicação confiável via **host name.**

**By doc: Differences between user-defined bridges and the default bridge.**

É uma comunicação bem mais estável que IP.

 <- Gerou um container e deu o nome de “ubuntu1”.

**Próximo passo criar nossa rede:**

**Comando: **

**Manda rodar o container criado dando o comando que ele vai ter a rede que criamos:**



Agora criou um novo container e colocou “minha bridge como rede”



Depois instala o app ping nos containers

Por ultimo deu um ping  (piadoca do carilho). Ping via host name.

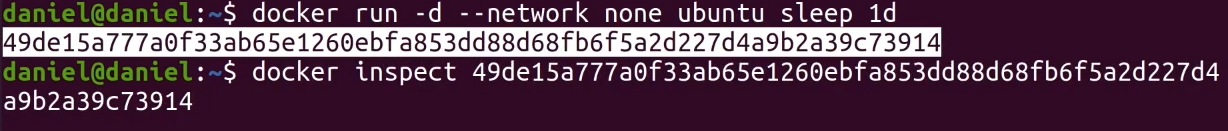
**Utilizando dive None**

Quaundo configuramos um container e como colocamos o drive de rede **NONE**, significa que este container não terá interface de rede, fica totalmente isolado a nível de rede.

**Utilizando dive HOST**

Interface de rede mais vinculada ao host, remove qualquer isolamento que existe entre host e container, no exemplo do professor o container que ele subiu estava com a porta 3000 para ser a porta de comunicação, usando o drive host colocou localhos:3000 e o Docker comunicou.

Se tivesse alguma outra aplicação utilizando a porta 3000 daria conflito.



Rede com drive “none “não tem qualquer interface de rede vinculado a ele.

Como que faz para ter uma interface de rede vinculada ao nosso host?



Aì é só colocar o numero da porta que foi vinculada no projeto

**Docker Compose**

O Docker Compose nada mais é do que uma ferramenta de coordenação de containers.

Docker composse é baseado a identação.

Docker Compose vai nos auxiliar a executar, a compor, como o nome diz, diversos containers em um mesmo ambiente, através de um único arquivo. Então vamos conseguir compor uma aplicação maior através dos nossos containers com o Docker Compose. O Docker Compose irá resolver o problema de executar múltiplos containers de uma só vez e de maneira coordenada, evitando executar cada comando de execução individualmente.

Quais alternativas abaixo contém parâmetros que podemos definir com o Docker Compose?

O nome do container é um parâmetro que pode ser definido com o Docker Compose.

A imagem é um parâmetro que pode ser definido com o Docker Compose.

A rede é um parâmetro que pode ser definido com o Docker Compose.,

**Criando um Docker Compose**

Para criar um Docker composse é necessário estar no diretório dos seus containers que serão incluídos no Compose.

Cria uma pasta chamada ymls.

Cria um arquivo “Docker-compose.yml”

Services: São os containers

Pode colocar o nome que quiser para os serviços e depois começa a inserir as peculiaridades dele, ex: qual a imagem do serviço, se a gente quer dar um nome para o container.

No meu arquivo Docker Compose tinha 2 erros:

1 ° - Erro de identação

2° Erro de build: ele estava procurando um diretório acima, e TINHA QUE COLOCAR NA RAIZ DO ARQUIVO.