**Docker** - Plataforma Open Source que permite desenvolver, publicar e executar aplicações separando a aplicação e infraestrutura, podendo ser utilizada usando contêiners e tornando a aplicação portátil. Com Docker podemos empacotar nossa aplicação juntamente com suas depêndencias e transportar ela para ser executada em qualquer outra máquina ou ambiente que rode o Docker.

**Contêiner** - Instância executável de uma aplicação/imagem, permite rodar várias aplicações de maneira isoladas, não utiliza SO próprio. Empacota uma apliacção e funciona de forma isolada do SO, com todas as dependências dentro de si e funciona independente do ambiente que está.

**Docker Contêiner**- Forma de empacotar o código e suas dependências, permitindo uma rápida execução em ambiente local, em VM ou nuvem.

**Imagem** - É imutável e possui o SO e todas as demais configuações necessárias para rodar a aplicação a qual esta imagem pertence.

**OBS**: Uma analogia entre **Imagem** e **Container**, umaimagem seria como uma classe (Java) e um contêiner seria o objeto dessa classe. Todas as alterações feita em um contêiner não afetam a imagem, pois ela é **imutável** e ao matar a instância de um contêiner o que foi gravado nele será perdido.

**Kubernetes** - Plataforma Open Source usada para gerenciar carga e trabalho, serviços distribudos em contêiners, implantar, orquestrar e escalar aplicações em contêiners, hospedar aplicações em nuvem que exigem escalabilidade rápida, verificar a integridade e autocorreção das apliações fazendo o reinício, replicação e esclonamento automático.

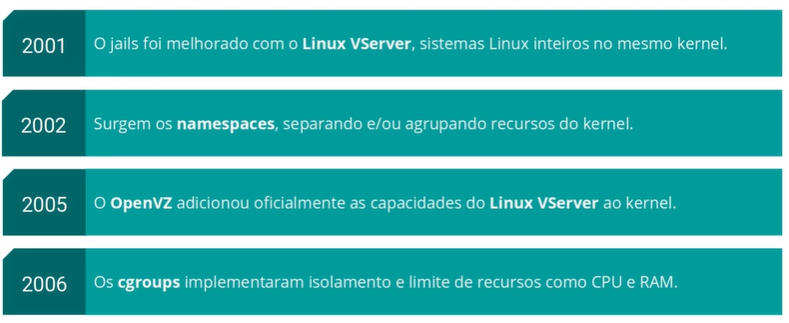
**POD** - Conjunto de 1 ou mais contêiners sendo a menor unidade de uma aplicação Kubernetes, os contêiners de um POD compartilham os mesmos recursos de computação, que são agrupados em clusters, proporionando um sistema distribuído de forma inteligente e eficiente.

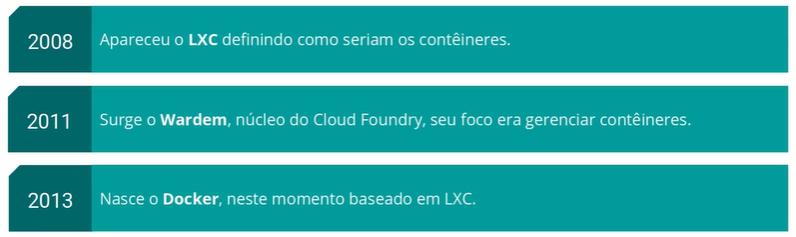
**Cluster** - Conexão entre dois ou mais computadores com o propósito de melhorar o desempenho dos sistemas na execução de diferentes tarefas.

**### CURSO CONTAINERS 4LINUX ###**

**Container** - Instância executável de uma aplicação/imagem, permite rodar várias aplicações de maneira isoladas, não utiliza SO próprio. Empacota uma apliacção e funciona de forma isolada do SO, com todas as dependências dentro de si e funciona independente do ambiente que está.

**Tecnologias envolvidas na ciração de um Contêiner**





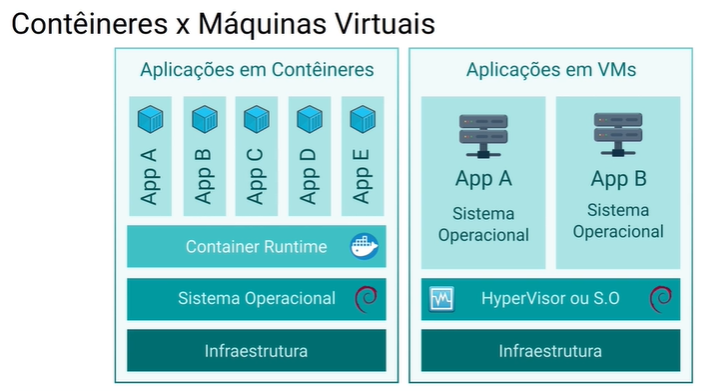
**Tecnologias Chaves**

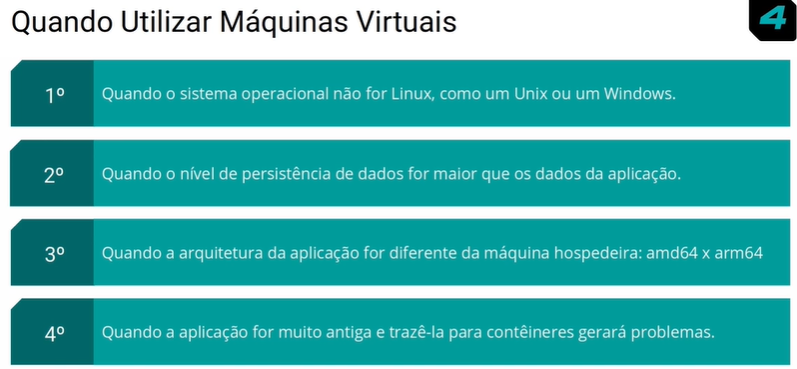
**Namespace:** Isolam os processos dos contêineres e também os pontos de montgem, diretórios, disco, redes, etc...

**Cgroups**: Limita o acesso a recursos da máquina como: CPU, memória e ráfego de rede.

**Union Mouting**: Sistem de arquivos que trabalhamem camadas que podem ser compartilhadas entre outros contêineres, podendo reduzir o tamanho da imagem. Exemplos de camadas: Union FS, Overlay2 e ZFS.

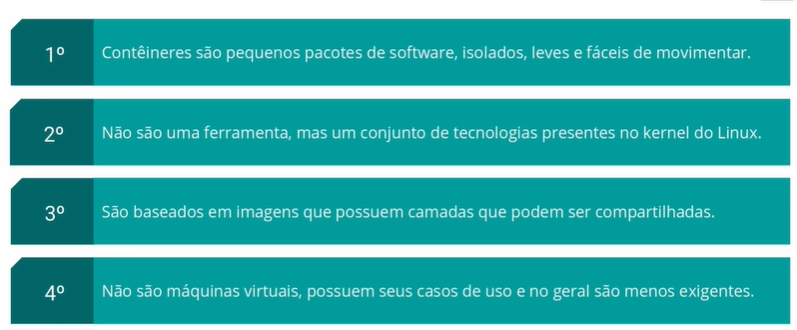
**Diferenças entre Contêineres x Máquina Virtual**







**RESUMO SOBRE CONTÊINERES**





**OPEN CONTAINER INITIATIVE (OCI)**

A OCI definiu os padrões dos contêineres, suas apis, imagens, inclusive junto com a própria Docker. Neste caso podemos usar outra tecnologia para criar contêineres como o Podman, pois os comandos e imagens são iguais.

**MODELO BIMODAL**

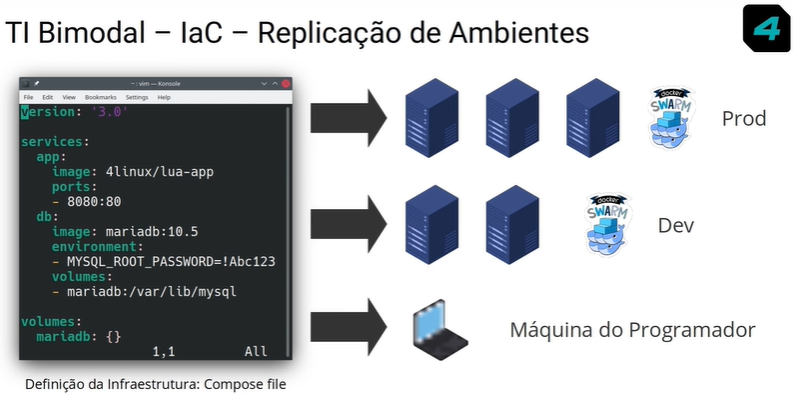
Consiste em um modelo de gestão que una tanto a TI tradicional (conhecida por suas metas bem definidas, estabilidade e eficiência) quanto uma TI mais experimental (focada na agilidade e inovação).

1 - Foco em previsibilidade e manutenção dos ambientes para o mundo digital (LEGADO)

2 - Foco em exploração/inovação, trabalham explorando novos ambientes, solucinando novos probemas e com incertezas (AMBIENTES MODERNOS)



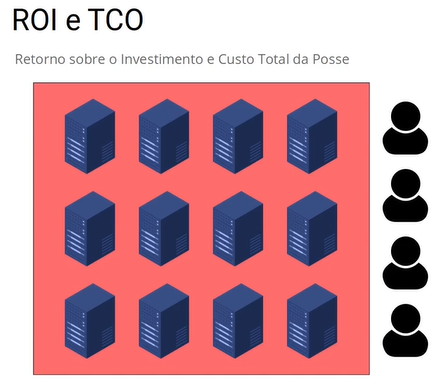




**SLA** - Service Level Agreement, nível de serviço prestado entre duas partes.

**SLO** - Service Level Object - Mensura os acordos firmados no SLA





**ORQUETRADOR** - Responsável por gerenciar e monitorar os contêineres, podendo restaurar os contêineres caso haja algum problema.

**SELF HELING** - Permte que uma aplicação tenha ciência sobre seu estado e seja reprovisionada caso ocorra algum problema.

**AUTOSCALING** - Aumenta ou diminui a quantidade de réplicas de um contêiner para suprir uma determinda demanda.

**BALANCEAMENTO DE CARGA** - Mesmo que a réplias aumentem ou diminuam, o orquestrador fornece um balanceamento entre as réplicas.

**SERVICE DISCOVERY** - Quando uma aplicação é adicionada ao cluster, eiste um mecanismo para encontrá-la de forma automática.

**ATUALIZAÇÃO CÍCLICA** - Uma atualização acontece em apenas uma das réplicas e o orquestrador não espalha em caso de problemas para outras réplicas.

Exemplos de orquestradores além do **Kubernetes**: Swarm Nomad.

**MICROSSERVIÇOS** - Pequenos serviços que se comunicam diretamente entre si, desenvolvidos de maneira independentes em seus respectivos funcionamentos, podendo ter linguagens e banco de dados diferentes. Temos aplicações resilientes, com fraca aoplação e componentes usando diversas e diferentes tecnologias, com fácil escalabilidade, devido a sua independencia entre os seus serviços.

**MONOLÍTICO** - Mais fácil de desenvolver, mais simples de testar e provisionar, porém qualquer alteração mínima exige um teste em toda a apliacação (ou pacote), recompilação e reprovisionamento completo. Toda implementação escalar toda a aplicação.

**SOAP (Simple Object Access Protocol) -** Protocolo de troca de informações baseado em XML e usado nas interações de serviços da web via HTTP ou RPC, as mensagens SOAP são tipicamente enviadas através de HTTP ou JMS.

**SOA (Arquitetura Orientada a Serviços)** - Design/Arquitetura que torna os componentes de software reutilizáveis por meio de interfaces de serviços com uma linguagem de comunicação comum em uma rede.

**ESB (Enterprise Service Bus)** - Usado na comunicação entre aplicações que interagem entre si no SOA, responsável por interceptar e traduzia mensagens entre as aplicações.

**SPOF (Single Point Failure)** - Parte da nossa aplicação que não possuem ou réplica ou alguma forma de se sobrepor as falhas.

**Sys Admin** - Para sua função é necessário saber criar contêiners, saber como funciona a comunicação entre contêiners via rede, levantar infra de testes e conhecer volumes. Aplicar coleta de métricas e monitorar a aplicação, propondo melhores práticas de comunicaçao e persistência, impondo limites de recursos.

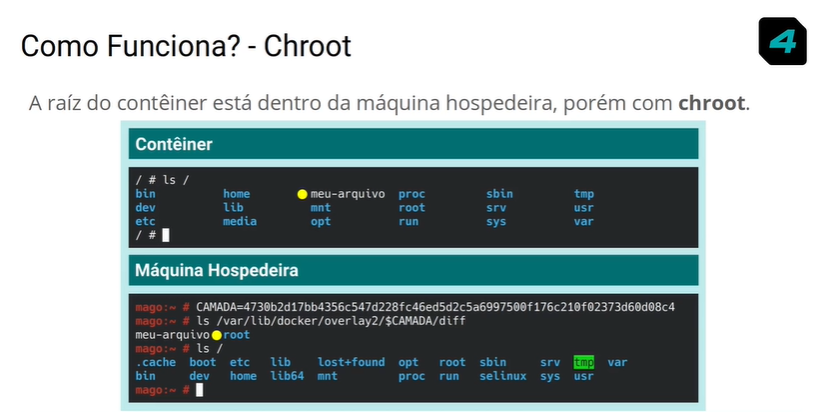
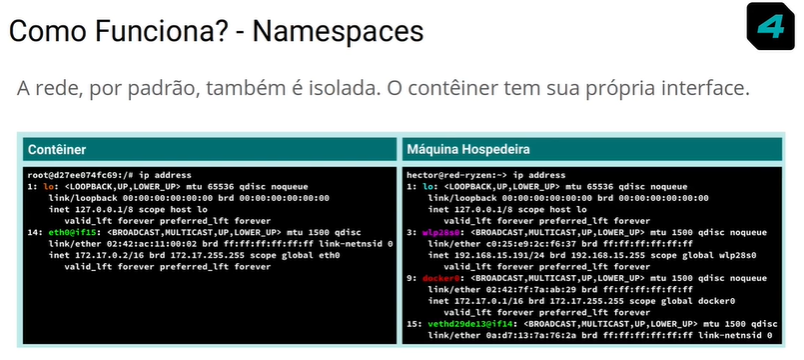
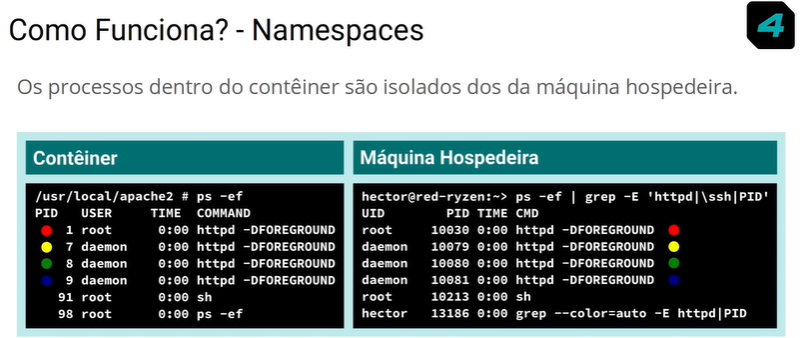
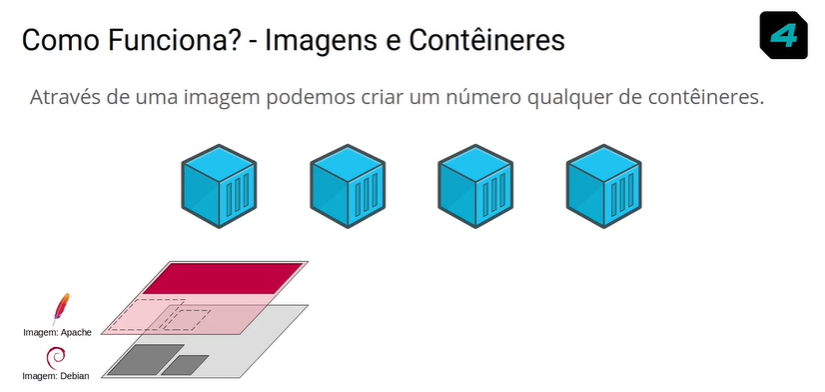
**Programador** - Criar e testar o código em suas respectivas imagens, utilizando os comandos Docker, lidar com infra local para testes, volumes distribuídos e cache.

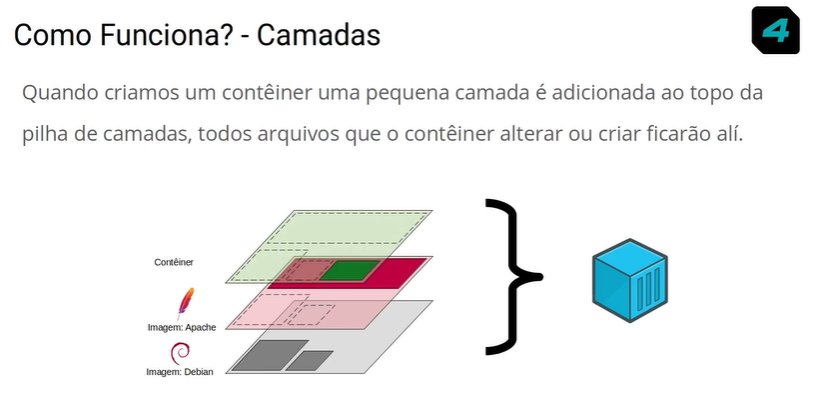
**DBA** - Os Operators relacionados aos BDs facilitam algumas tarefas como backups, replicação e alta disponibilidade.

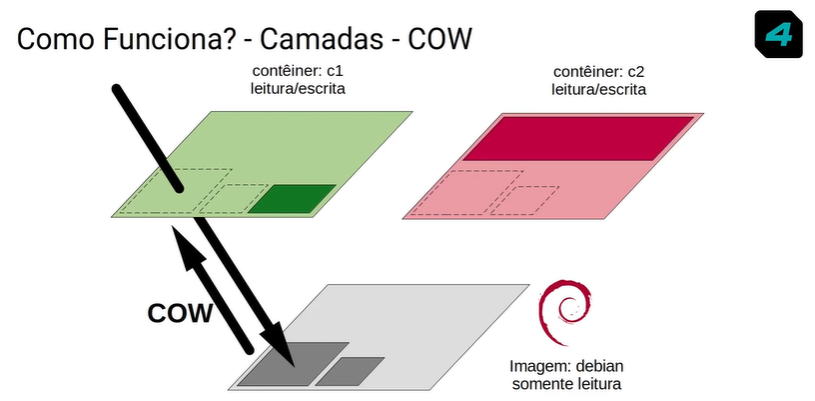
**DOCKER** - Ferramenta de desenvolvimento e plataforma de código aberto, responsável por utilizar virtualização em nível de SO para criar e gerenciar aplicações em formato de contêineres. Traz facilidade em empacotar e compartilhar aplicações.

**Imagem** - Contém a aplicação, suas dependências, libs, binários e outros arquivos que a aplicação necessita para o seu funcionamento. A imagem criada pós desenvolvimento é a mesma usada no ambiente de produção. As imagens são imutáveis e podem possuir diversa camadas, as camadas podem ser compartilhadas entre si, onde uma imagem pode utilizar a camada de outra imagem como base.

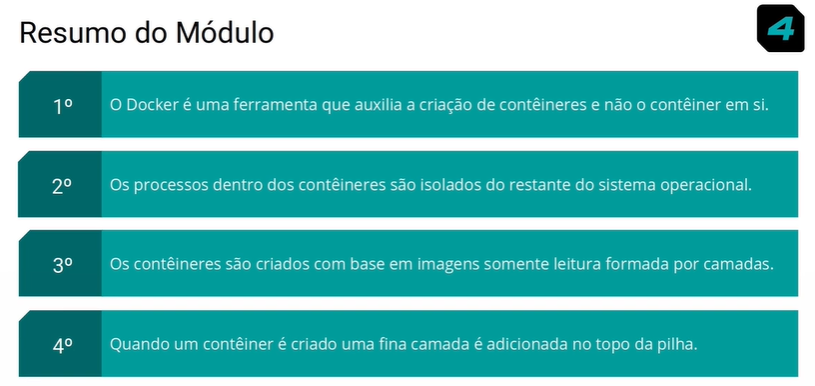
**OBS**: Uma imagem pode ter uma ou mais camadas, na maioria das vezes possuem mais de uma.











**COMANDOS DOCKER**

**Segue a lista de comandos docker e sua utilidade**

**docker attach** – Acessar dentro do container e trabalhar a partir dele.

**docker build** – A partir de instruções de um arquivo Dockerfile eu possa criar uma imagem.

**docker commit** – Cria uma imagem a partir de um container.

**docker cp** – Copia arquivos ou diretórios do container para o host.

**docker create** – Cria um novo container.

**docker diff** – Exibe as alterações feitas no filesystem do container.

**docker events** – Exibe os eventos do container em tempo real.

**docker exec** – Executa uma instrução dentro do container que está rodando sem precisar atachar nele.

**docker export** – Exporta um container para um arquivo .tar.

**docker history** – Exibe o histórico de comandos que foram executados dentro do container.

**docker images** – Lista as imagens disponíveis no host.

**docker image build –** Usado para cria uma imagem (deve receber como parâmetro o nome da imagem – Ex: docker image build –tag flask)

**docker image push** – Subir para o registry a imagem criada com o build

**docker import** – Importa uma imagem .tar para o host.

**docker info** – Exibe as informações sobre o host.

**docker inspect** – Exibe r o json com todas as configurações do container.

**docker kill** – Da Poweroff no container.

**docker load** – Carrega a imagem de um arquivo .tar.

**docker login** – Registra ou faz o login em um servidor de registry.

**docker logout** – Faz o logout de um servidor de registry.

**docker logs** – Exibe os logs de um container.

**docker port** – Abre uma porta do host e do container.

**docker network** – Gerenciamento das redes do Docker.

**docker node** – Gerenciamento dos nodes do Docker Swarm.

**docker pause** – Pausa o container.

**docker port** – Lista as portas mapeadas de um container.

**docker ps** – Lista todos os containers.

**docker pul**l – Faz o pull de uma imagem a partir de um servidor de registry.

**docker push** – Faz o push de uma imagem a partir de um servidor de registry.

**docker rename** – Renomeia um container existente.

**docker restart** – Restarta um container que está rodando ou parado.

**docker rm** – Remove um ou mais containeres.

**docker rmi** – Remove uma ou mais imagens.

**docker run** – Executa um comando em um novo container.

**docker save** – Salva a imagem em um arquivo .tar.

**docker search** – Procura por uma imagem no Docker Hub.

**docker service** – Gernciamento dos serviços do Docker.

**docker start** – Inicia um container que esteja parado.

**docker stats** – Exibe informações de uso de CPU, memória e rede.

**docker stop** – Para um container que esteja rodando.

**docker swarm** – Clusterização das aplicações em uma orquestração de vários containers, aplicações junto.

**docker tag** – Coloca tag em uma imagem para o repositorio.

**docker top** – Exibe os processos rodando em um container.

**docker unpause** – Inicia um container que está em pause.

**docker update** – Atualiza a configuração de um ou mais containers.

**docker version** – Exibe as versões de API, Client e Server do host.

**docker volume** – Gerenciamento dos volumes no Docker.

**docker wait** – Aguarda o retorno da execução de um container para iniciar esse container.

|  |  |
| --- | --- |
| [docker container attach](https://docs.docker.com/engine/reference/commandline/container_attach/) | Anexar fluxos de entrada, saída e erro locais padrão a um contêiner em execução |
| [docker container commit](https://docs.docker.com/engine/reference/commandline/container_commit/) | Cria uma nova imagem a partir das alterações de um container |
| [docker container cp](https://docs.docker.com/engine/reference/commandline/container_cp/) | Copia arquivos/pastas entre um container e o sistema de arquivos local |
| [docker container create](https://docs.docker.com/engine/reference/commandline/container_create/) | Criar um novo container |
| [docker container diff](https://docs.docker.com/engine/reference/commandline/container_diff/) | Inspecionar alterações em arquivos ou diretórios no sistema de arquivos de um contêiner |
| [docker container exec](https://docs.docker.com/engine/reference/commandline/container_exec/) | Execute um comando em um contêiner em execução |
| [docker container export](https://docs.docker.com/engine/reference/commandline/container_export/) | Exporta o sistema de arquivos de um container como um arquivo tar |
| [docker container inspect](https://docs.docker.com/engine/reference/commandline/container_inspect/) | Exibir informações detalhadas sobre um ou mais contêineres (informar o nome do container como parâmetro) |
| [docker container kill](https://docs.docker.com/engine/reference/commandline/container_kill/) | Mate um ou mais containers em execução |
| [docker container logs](https://docs.docker.com/engine/reference/commandline/container_logs/) | Buscar os logs de um contêiner |
| [docker container ls](https://docs.docker.com/engine/reference/commandline/container_ls/) | Listar contêineres |
| [docker container ls -a](https://docs.docker.com/engine/reference/commandline/image_ls/) | Mostrar todas as imagens (o padrão oculta imagens intermediárias) |
| [docker container pause](https://docs.docker.com/engine/reference/commandline/container_pause/) | Pausar todos os processos em um ou mais containers |
| [docker container port](https://docs.docker.com/engine/reference/commandline/container_port/) | Listar mapeamentos de portas ou um mapeamento específico para o contêiner |
| [docker container prune](https://docs.docker.com/engine/reference/commandline/container_prune/) | Remover todos os containers parados |
| [docker container rename](https://docs.docker.com/engine/reference/commandline/container_rename/) | Renomear um contêiner |
| [docker container restart](https://docs.docker.com/engine/reference/commandline/container_restart/) | Reinicie um ou mais contêineres |
| [docker container rm](https://docs.docker.com/engine/reference/commandline/container_rm/) | Remova um ou mais containers (para remover um container que esteja rodando usar o parâmetro -f após o rm) |
| [docker container run](https://docs.docker.com/engine/reference/commandline/container_run/) | Executar um comando em um novo container |
| [docker container start](https://docs.docker.com/engine/reference/commandline/container_start/) | Iniciar um ou mais containers parados |
| [docker container stats](https://docs.docker.com/engine/reference/commandline/container_stats/) | Exibir uma transmissão ao vivo das estatísticas de uso de recursos do(s) contêiner(es) |
| [docker container stop](https://docs.docker.com/engine/reference/commandline/container_stop/) | Parar um ou mais containers em execução |
| [docker container top](https://docs.docker.com/engine/reference/commandline/container_top/) | Exibir os processos em execução de um container |
| [docker container unpause](https://docs.docker.com/engine/reference/commandline/container_unpause/) | Retome todos os processos dentro de um ou mais containers |
| [docker container update](https://docs.docker.com/engine/reference/commandline/container_update/) | Atualizar a configuração de um ou mais contêineres |
| [docker container wait](https://docs.docker.com/engine/reference/commandline/container_wait/) | Bloqueie até que um ou mais containers parem, então imprima seus códigos de saída |
| [docker container --name](https://docs.docker.com/engine/reference/run/#name---name) | Aplica um (nome) identificador para o container (obs: após o name podemos passar o nome da imagem |
|  |  |

**Máscara da rede do Docker -** 172.17.0.2/16 a 172.17.0.254

**curl -** Comando usado para fazer requisições HTTP via terminal

**Alpine Linux -** Distro voltada para segurança, muito usado com containeres.

**TAG** - As tags Docker transmitem informações úteis sobre uma versão / variante de imagem específica. Eles são apelidos para o ID de sua imagem, que geralmente se parece com isto: f1477ec11d12. É apenas uma forma de se referir à sua imagem, é uma abstração para criar unidade dentro do conjunto de imagens definidas no **“repositório”**.

**DockerHub** - O container é uma imagem que eu empacotei tudo que minha aplicação precisa para rodar. E existe um lugar público chamado Docker Hub onde várias empresas e pessoas publicam imagens pré-compiladas de soluções. Então, lá você vai poder, por exemplo, encontrar uma imagem pronta para MySQL, Java e diversas outras soluções. Em outras palavras: o Docker Hub é um repositório público onde empresas podem publicar suas soluções em forma de container”.

***CASO DE USO*** *- Devemos nos preocupar com o tamanho da imagem usada com o Docker, como exemplo, em um ambiente multi-cloud gerenciado via cluster Kubernetes o tráfego de rede que entra e sai do container e cobrado e diminuindo o tamanho da imagem temos um custo menor de cloud, se estivermos utilizando CI/CD como o Jenkins, por exemplo, com imagens menores facilita o build. Uma imagem menor possui menos softwares embarcados nela, tornando-a mais segura, como a Distro do Alpine, que nos fazer ter mais segurança e economia de banda.*

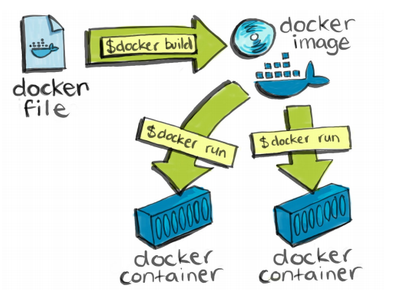
**Comando para fechar um container sem encerrar o processo**: control + p + q

**Para iniciar um container e não deixar o processo preso no terminal usamos o parâmetro** (Ex: docker container run --tty --interactive --detach alpine)

**Para iniciar um container e rodar o terminal dentro dele de maneira interativa usamos os parâmetros: --tty --interactive** (Ex: docker container run --tty --interactive alpine)

***CASO DE USO*** *- Ao criarmos imagem é mais simples e mais organizado se nos basearmos em uma imagem "base".*

**Tempo de Vida Contâiner**: O tempo de vida de um contâiner é dependente da execução do processo principal, se em seu contâiner roda um simples hello world exibido ao usuário, após rodar o comando o container é encerrado.



**IMAGEM VS. CONTÊINER**

A imagem é um arquivo e o contêiner é um processo. Da perspectiva do kernel Linux, um contêiner é um processo com restrições. No entanto, ao invés de executar um único arquivo binário, um contêiner executa uma imagem. Uma imagem é um pacote de sistema de arquivos que contém todas as dependências necessárias para executar um processo: arquivos de biblioteca, arquivos no sistema de arquivos, pacotes instalados, recursos disponíveis, processos em execução e módulos do kernel.

**VOLUMES** - A principal função do volume é persistir os dados, quando você escreve em um volume aquele dado continua lá, independentemente do estado do *container*.

* O volume é inicializado quando o *container* é criado.
* Caso ocorra de já haver dados no diretório em que você está montando como volume, ou seja, se o diretório já existe e está "populado" na imagem base, aqueles dados serão copiados para o volume.
* Um volume pode ser reusado e compartilhado entre *containers*.
* Alterações em um volume são feitas diretamente no volume.
* Alterações em um volume não irão com a imagem quando você fizer uma cópia ou *snapshot* de um *container*.
* Volumes continuam a existir mesmo se você deletar o *container*.

**Comando para criar dentro o container um volume com o conteúdo de um diretório:**



**Comando usado para criar um container baseado no mariadb com variáveis de ambiente:** 

**Comando para acompanhar os logs do container**: docker container logs (nome do container) -f (significa follow que vai seguir os logs do container)

**Comando para criar um container com volume gerenciado pelo Docker:**



**Comando para listar volumes**: docker volume ls (ou o ls -a)

**Comando para redirecionar requisições de uma porta para outra:**

docker run –publish (-p) 8080:80 mysql

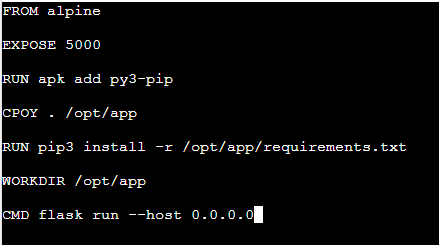
**DockerFile** – Dockerfile, que nada mais é do que um arquivo de definição onde é possível realizar ou preparar todo ambiente a partir de um script de execução. Em resumo, o Dockerfile é um arquivo texto com instruções, comandos e passos que você executaria manualmente, é a descrição do que nossa aplicação possui (dependências, versões de libs, porta de execução da aplicação e etc)

**Dockerfile – Instruções Disponíveis:**

* **FROM:** Informa a partir de qual imagem será gerada a nova imagem, lembrando que em poucos casos (Veremos em posts futuros), uma imagem será gerada se uma imagem base;
* **MAINTAINER:** Campo opcional, que informa o nome do mantenedor da nova imagem;
* **RUN:** Especifica que o argumento seguinte será executado, ou seja, realiza a execução de um comando;
* **CMD:** Define um comando a ser executado quando um container baseado nessa imagem for iniciado, esse parâmetro pode ser sobrescrito caso o container seja iniciado utilizando alguma informação de comando, como: **docker run -d imagem comando**, neste caso o CMD da imagem será sobrescrito pelo comando informado;
* **LABEL:** Adiciona metadados a uma imagem, informações adicionais que servirão para identificar versão, tipo de licença, ou host, lembrando que a cada nova instrução LABEL é criada uma nova layer, o Docker recomenda que você não use muitas LABEL. É possível realizar filtragens posteriormente utilizando essas LABEL.
* **EXPOSE:** Expõem uma ou mais portas, isso quer dizer que o container quando iniciado poderá ser acessível através dessas portas;
* **ENV:** Instrução que cria e atribui um valor para uma variável dentro da imagem, isso é útil para realizar a instalação de alguma aplicação ou configurar um ambiente inteiro.
* **ADD:** Adiciona arquivos locais ou que estejam em uma url, para dentro da imagem.
* **COPY:** Copia arquivos ou diretórios locais para dentro da imagem.
* **ENTRYPOINT:** Informa qual comando será executado quando um container for iniciado utilizando esta imagem, diferentemente do CMD, o ENTRYPOINT não é sobrescrito, isso quer dizer que este comando será sempre executado.
* **VOLUME:** Mapeia um diretório do host para ser acessível pelo container;
* **USER:** Define com qual usuário serão executadas as instruções durante a geração da imagem;
* **WORKDIR:** Define qual será o diretório de trabalho (lugar onde serão copiados os arquivos, e criadas novas pastas);
* **ONBUILD:** Define algumas instruções que podem ser realizadas quando alguma determinada ação for executada, é basicamente como uma trigger.

**Exemplo de um Dockerfile**

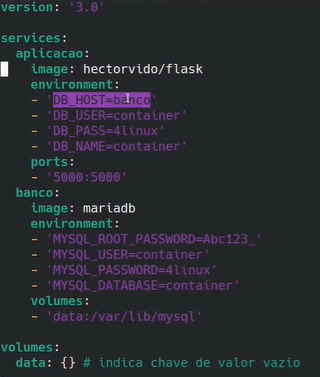
**OBS: o comando correto é COPY na linha 4 do arquivo abaixobb**



**Docker registry** - É um repositório de imagens para que você consiga subir (push) e baixar (pull) imagens dele. Exemplos de Registry: Suse Portus, Harbor, Nexus Sonatype e etc...

**Docker Compose** - É uma ferramenta para a criação e execução de múltiplos containers de aplicação. Com o Compose, você usará um arquivo do tipo yaml para definir como será o ambiente de sua aplicação e usando um comando que você criará e iniciará todos os únicos serviços.

**Exemplo de um docker-compose:**



**Comando para listar os serviços no docker-compose:** docker-compose os

**Comando para iniciar os serviços no yaml do docker-compose:** docker-compose up

**Comando para ver os logs no yaml do docker-compose:** docker-compose logs -f

**Comando para parar os containeres do compose:** docker-compose down