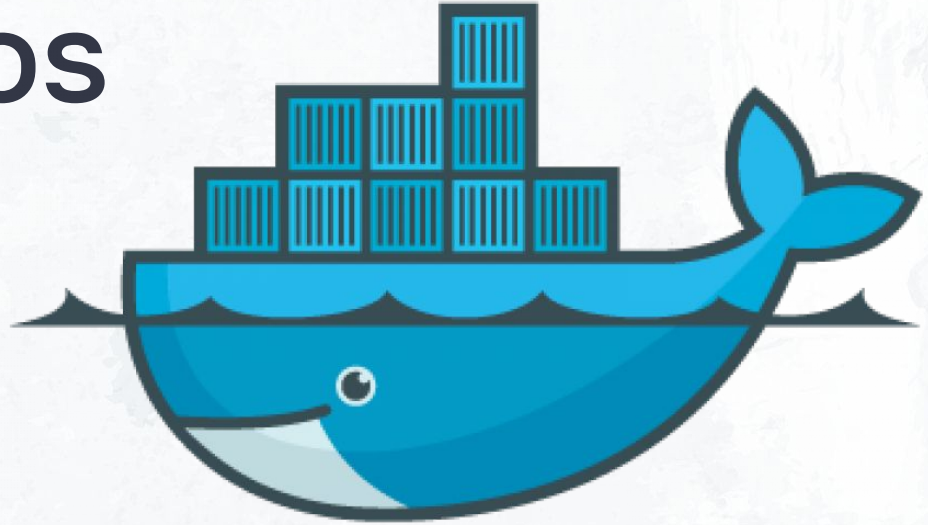
Fundamentos do Docker



docker

Índice

Material de Apoio

Baixe o ‘**material de apoio**’ e este slide no **git:**

https://github.com/nerdsufc/curso-docker.git

00

O que é o Docker e quais problemas ele soluciona?

Contextualizando

A maioria dos aplicativos modernos utilizam uma variedade de tecnologias para criar funcionalidades completas, o que acarreta em **custos** adicionais, tais como **implantação, manutenção e consumo de recursos computacionais**.

Problemas

Implantação

- Qual **sistema operacional ele roda**;
- Quais recursos ele precisa, de **que outro software ele depende**;
- Algum outro **software já instalado pode interferir na instalação**.

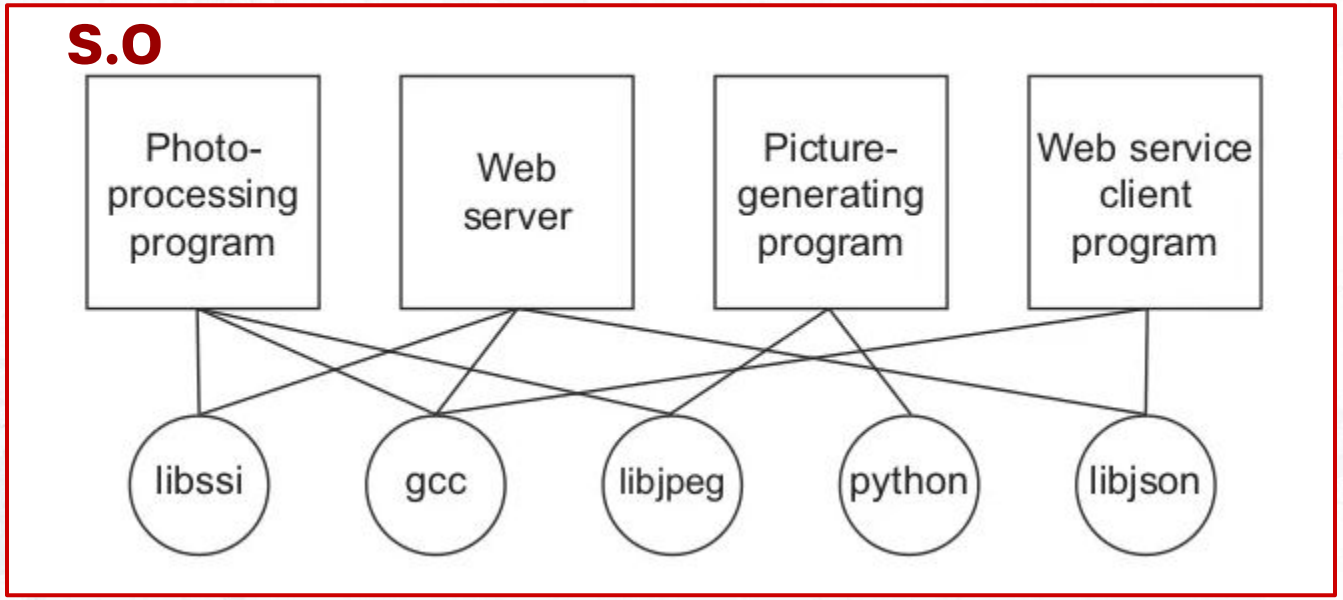
Consumo excessivo de recurso computacional

- Digamos que, por qualquer motivo, o **consumo** de memória de um aplicativo comece a crescer e, depois de um tempo, ele esgote toda a memória. Fatalmente **outras aplicações/serviços também falharão**.

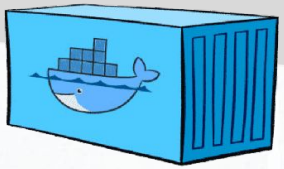
Manutenção

- Os servidores têm **vários aplicativos** em execução **consumindo recursos computacionais**.
- Se um aplicativo precisar de **atualização**, mas outro app for executado em uma **versão mais antiga da dependência**?
- Quando queremos remover um **serviço**, precisamos nos lembrar de todas as **alterações/configurações** que tivemos que fazer e **desfazê-las**.

Problemas


S.O
Photo-processing program
Web server
Picture-generating program
Web service client program
libssi
gcc
libjpeg
python
libjson

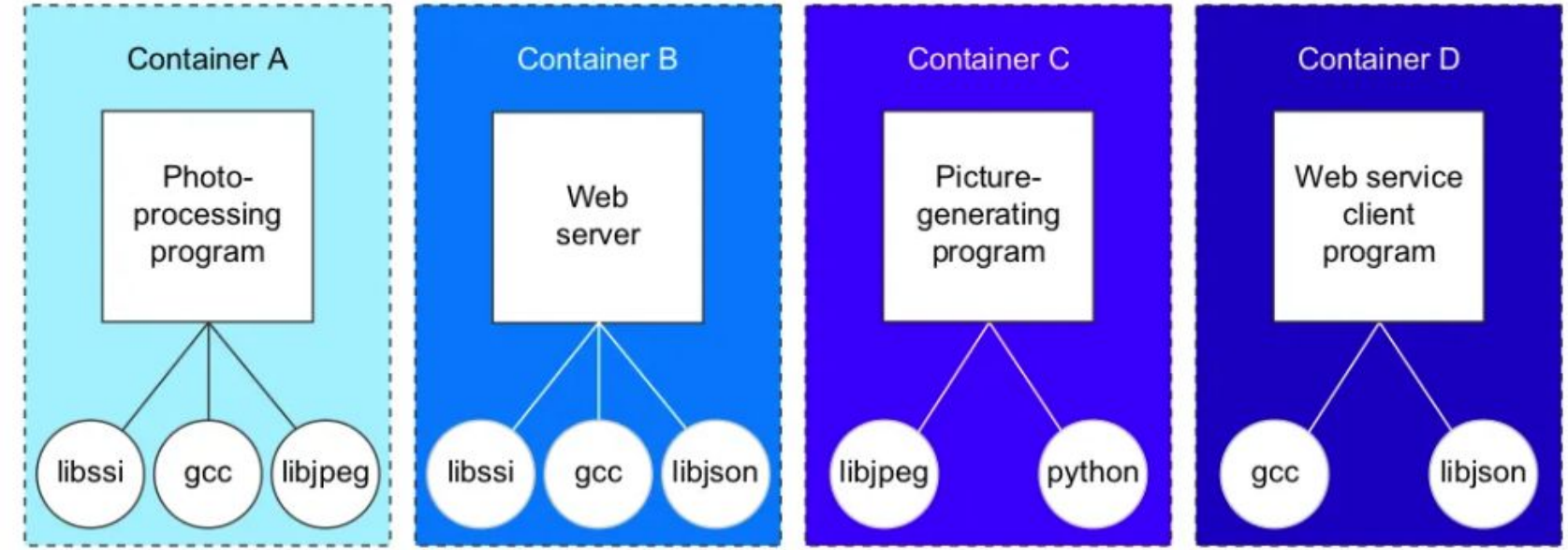
O que é Docker



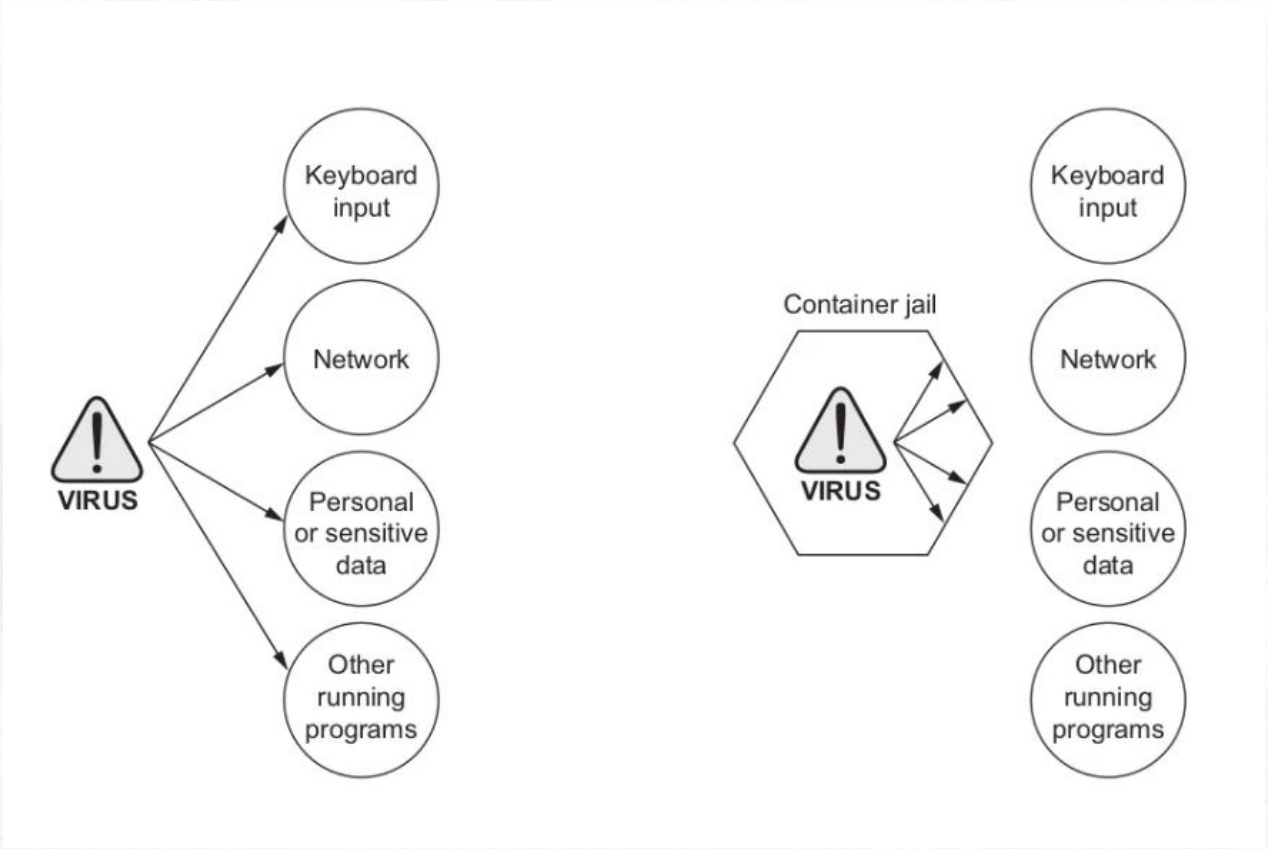
O Docker é uma plataforma open source que facilita a criação e administração de ambientes isolados. Ele possibilita o **empacotamento de uma aplicação ou ambiente dentro de um container**.

Os containers são isolados a nível de **disco, memória, processamento e rede**. Essa separação permite grande flexibilidade, onde ambientes distintos podem coexistir no mesmo host. Containers também carregam **todos os arquivos necessários** (configuração, biblioteca e afins) para **execução completamente isolada**.

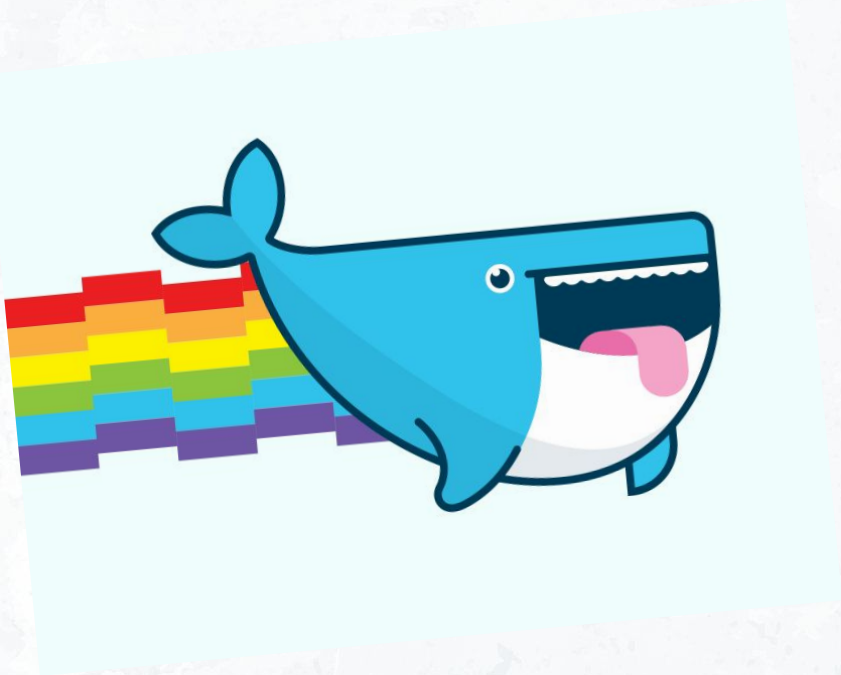
Soluções


Container A
Photo-processing program
libssi
gcc
libjpeg
Container B
Web server
libssi
gcc
libjson
Container C
Picture-generating program
libjpeg
python
Container D
Web service client program
gcc
libjson

Soluções


Keyboard input
Network
Personal or sensitive data
Other running programs
VIRUS
Container jail
VIRUS
Keyboard input
Network
Personal or sensitive data
Other running programs

Vantagens de usar Docker

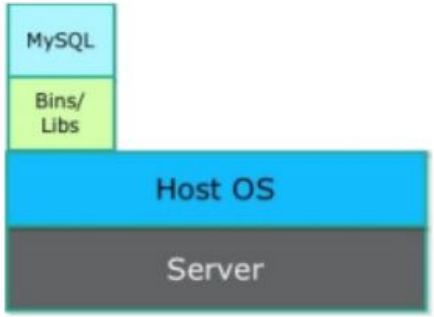


- Construa uma vez e **execute em qualquer lugar**;
- Ambiente limpo, leve e isolado. Os contêineres **levam apenas o necessário para sua finalidade.** O isolamento garante **segurança e uma melhor gestão da aplicação**;
- Alinhamento com a arquitetura de **microsserviços** e **automatização**;
- Não se preocupe com falta de **dependências, pacotes ou versões incompatíveis**;
- Ambientes de desenvolvimento consistentes para todo time. Os desenvolvedores usam o mesmo SO, bibliotecas, o mesmo tempo de execução, **independentemente do host**;
- Implantação é fácil. O **comportamento do contêiner** executado em seu ambiente de desenvolvimento, **será o mesmo em produção**.

Bare Metal x VM x Contêiner

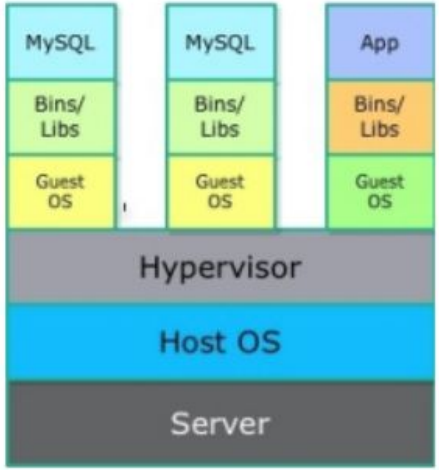
Bare Metal

Openstack ironic



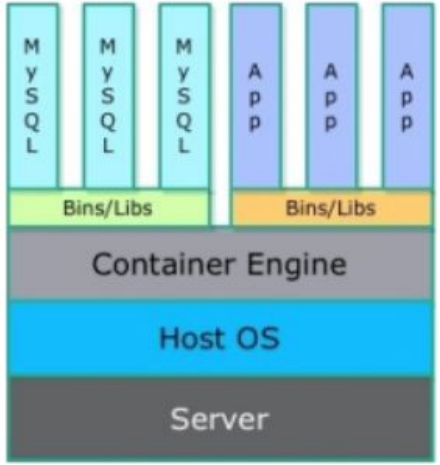
VM Based Host

Standard nova driver



Container

Nova-docker driver



Mão na Massa!

- Isolamento de processos, do host e do container;
- Destruindo um contêiner e subindo outro novinho.

01

Conceitos fundamentais da tecnologia de containers

Arquitetura Docker

ARQUITETURA DOCKER

- DOCKER **ENGINE**;
- DOCKER **CLIENT**;
- DOCKER **OBJECTS**;
- DOCKER **REGISTRY**;

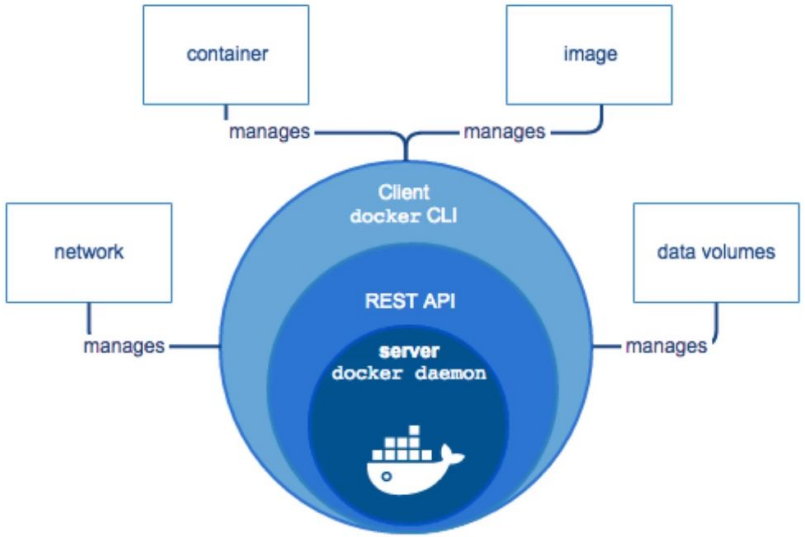
Mão na massa:
Instalando o docker

```
https://docs.docker.com/engine/install/
```

Arquitetura Docker

DOCKER ENGINE

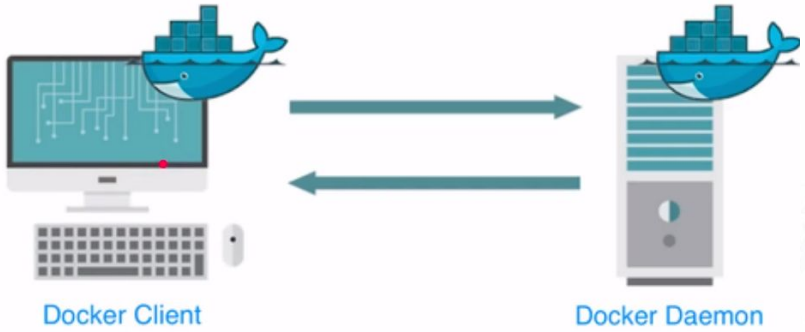
- **DESKTOP:** Mac OS, Windows 10
- **SERVER:** Various Linux distributions e Windows Server 2016
- **CLOUD:** Amazon Web Services, Google Compute Platform, Microsoft Azure, IBM Cloud, DigitalOcean e mais



Arquitetura Docker

DOCKER CLIENT

- O Client recebe as entradas do usuário (CLI) e as envia para a Engine
  - docker build
  - docker pull
  - docker run
- Client e a Engine podem ser *executados* em **hosts iguais** ou **diferentes**



Arquitetura Docker

DOCKER OBJECTS

IMAGES (*"PRODUÇÃO"*)

- Template (*read-only*) usado para criar containers - core central;
- Construído pela comunidade, por mantenedores ou por você;
- Armazenado num Docker "Registry" público ou local;

CONTAINERS (*"CONSUMO"*)

- Isolamento da aplicação e de recursos;
- Efêmeros;
- Contém *o necessário* para executar uma aplicação;
- Baseado nas IMAGES;

Arquitetura Docker

DOCKER OBJECTS

NETWORKING

- Capacidade dos containers *comunicarem-se* entre si;
- Abstração da complexidade de rede através de *plugins drivers*;
- User-defined networks: *bridge, overlay* e *macvlan*;

STORAGE

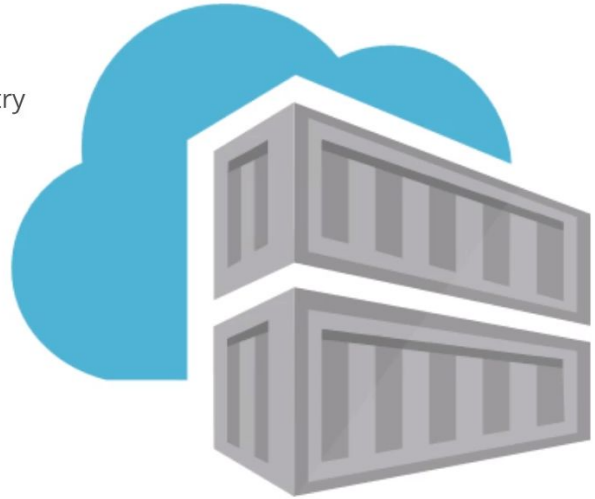
- Persistência de dados (container elemento *volátil*);
- Storage drivers: *docker volume, bind mount* e *tmpfs mounts (Linux)*;

Arquitetura Docker

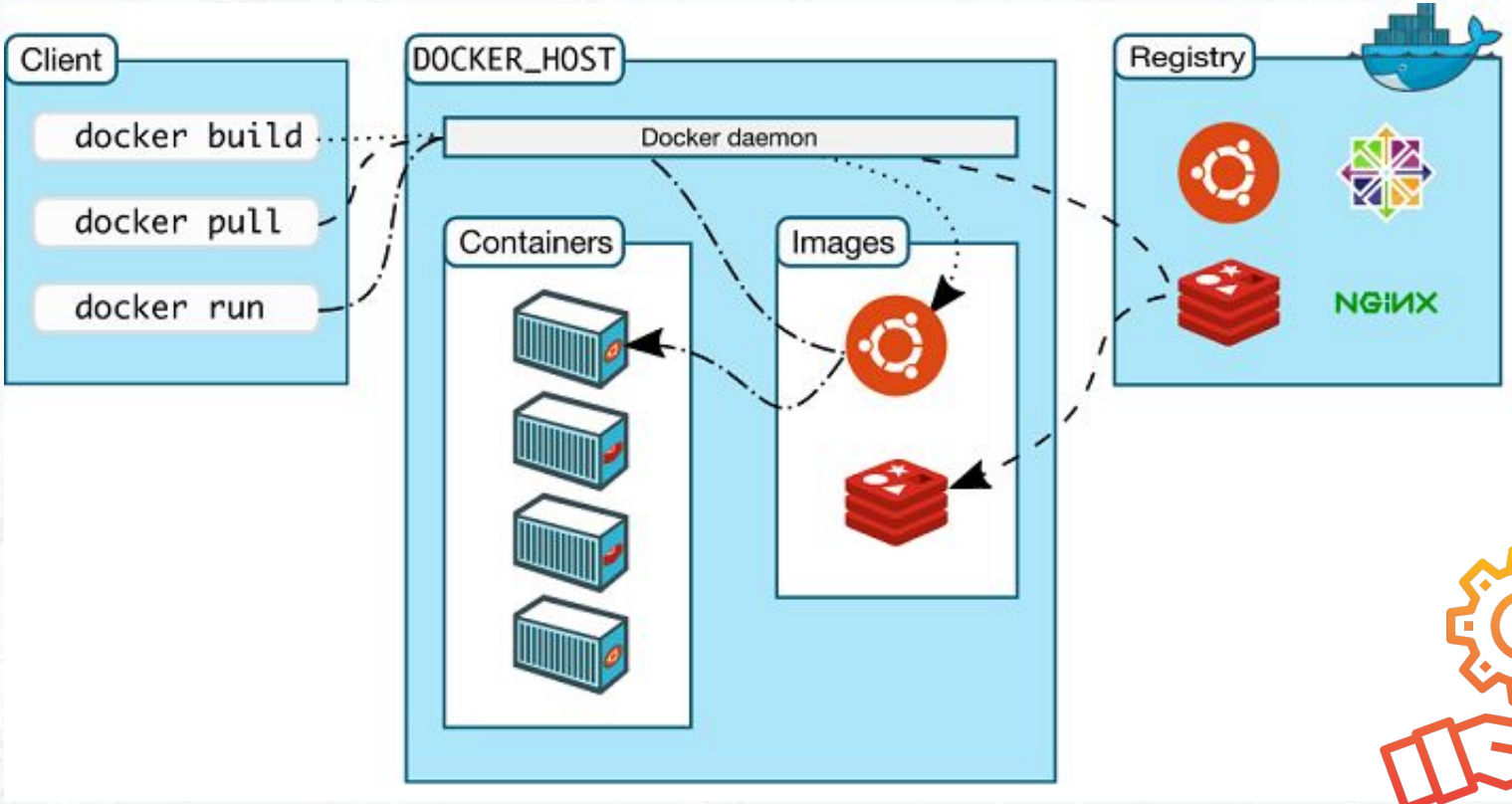
DOCKER REGISTRY

REPOSITÓRIO

- Serviço que gerencia o *armazenamento* das IMAGES;
- **Público**:
  - Docker Hub, Docker Cloud, Quay.io e Google Container Registry
- **Privado**:
  - Repositório local



Mão na Massa!


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

02

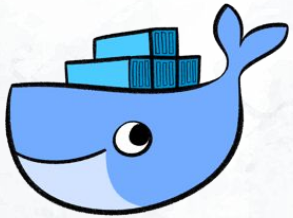
Produção e consumo

O que são Imagens?

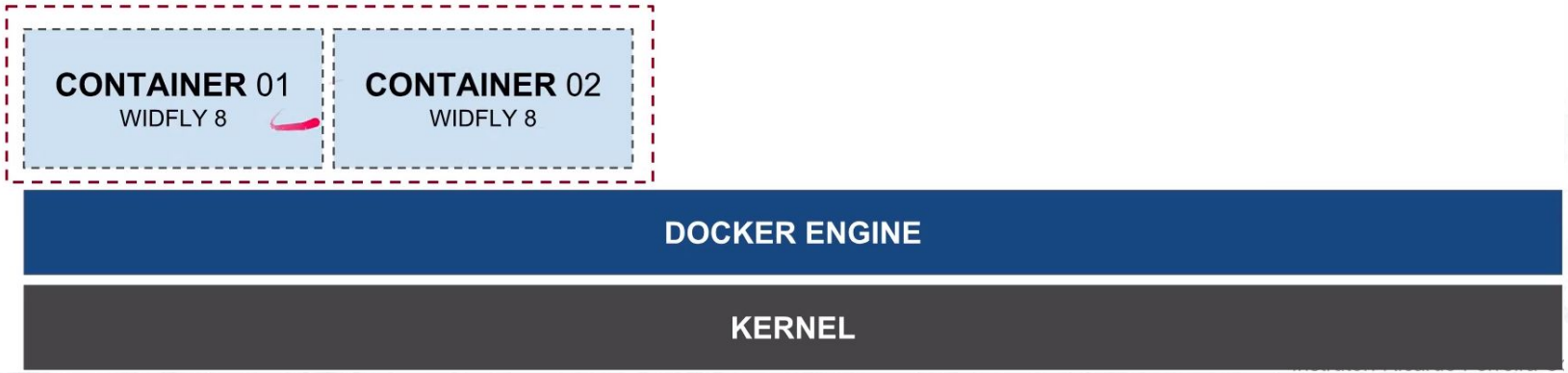
Uma imagem é um pacote autossuficiente que contém tudo o que é necessário para executar um software, incluindo **o código**, **as bibliotecas**, **as dependências** e **as configurações**.

Elas são criadas a partir de um conjunto de instruções chamado **Dockerfile**(receita), que descreve como construir a imagem.

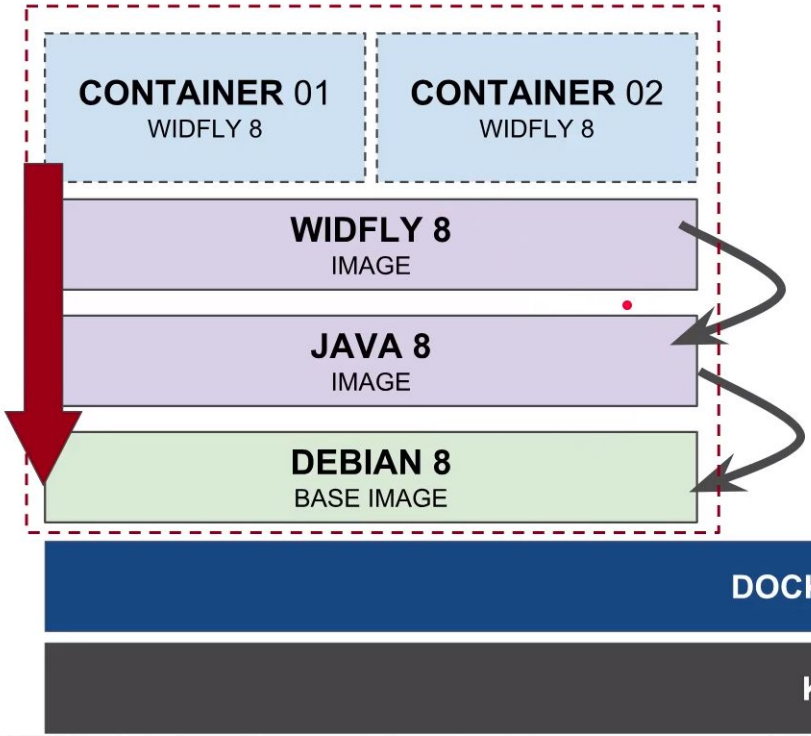
Produção



ESTRUTURA DOCKER IMAGE



ESTRUTURA DOCKER IMAGE



Uma **imagem** é **composta por uma estrutura de camadas empilhadas**, onde cada camada contém alterações em relação à camada anterior, incluindo **instruções e pacotes de suporte à aplicação**.

O **contêiner** é o **elemento principal da imagem**, atuando na camada de escrita (writable). As demais camadas **fornecem suporte à camada de escrita** e estão no modo somente leitura (read-only).

O que são contêineres?

Um **contêiner é uma instância em execução de uma imagem**. Ele é **isolado** e **encapsula o ambiente necessário para executar um aplicativo**, incluindo o sistema operacional, as bibliotecas e as configurações específicas.

Eles são leves, independentes e **compartilham o kernel do sistema operacional** do host, permitindo que os aplicativos sejam executados de forma consistente em diferentes ambientes.

Consumo

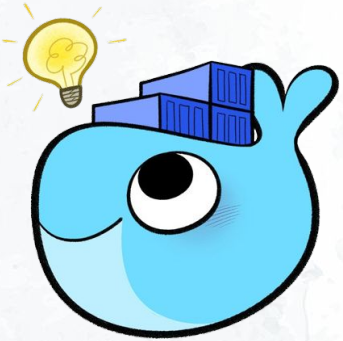
Mão na Massa!

- Vamos usar os comandos do docker para entender na prática sobre imagens e contêineres

Projeto:

subindo um servidor **HTTP**, e um **website**

NGINX



Redes com Docker

O Docker oferece recursos abrangentes para gerenciar redes em ambientes de contêineres, permitindo **a criação e configuração de redes internas e externas**, bem como **a comunicação entre contêineres e hosts**.

Isso possibilita a construção de ambientes de aplicativos complexos e distribuídos usando contêineres isolados e conectados em uma rede.

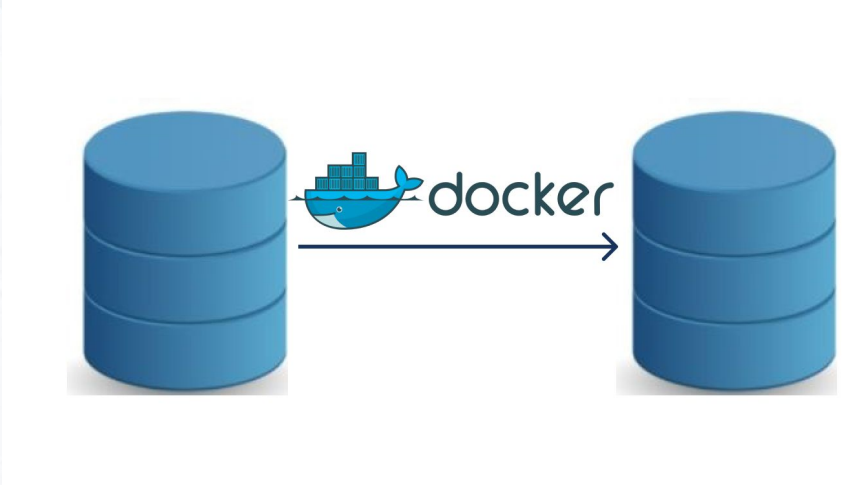
Mão na Massa!

- Vamos criar um contêiner com serviço mysql, inspecionar o contêiner e verificar o endereço IP e acessá-lo via endereço de rede.

Volumes

Volumes são mecanismos que permitem que os dados sejam **compartilhados e persistam entre os contêineres e o host**.

Eles são usados para armazenar arquivos, bancos de dados, logs ou qualquer outro tipo de dado que precise ser compartilhado ou persistido além da vida útil do contêiner.



Mão na Massa!

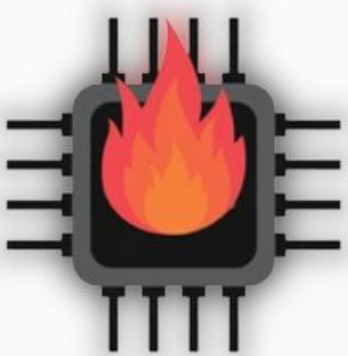
- Vamos criar uma pasta chamada **'dados'** na pasta home do nosso usuário. Em seguida, iremos iniciar um contêiner e persistir os dados internos no diretório **dados** no host.

- Após isso, iremos remover o contêiner inicial e iniciar outro contêiner para acessar os mesmos dados persistidos.

03

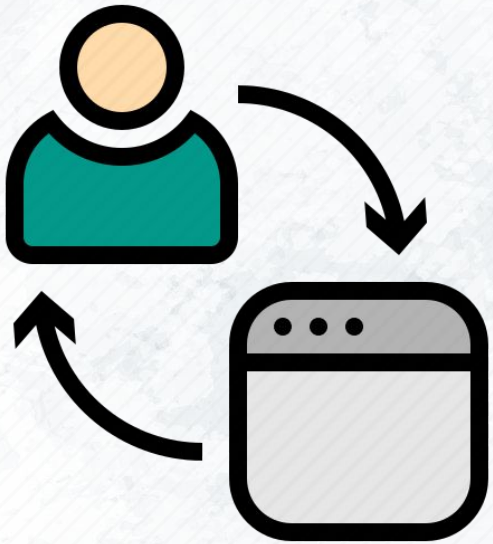
Administração de containers

Controle de recursos



O controle de recursos é essencial em contêineres para evitar problemas de desempenho e instabilidade. Ao limitar o uso de CPU, memória e armazenamento, é possível evitar que um contêiner monopolize todos os recursos, prejudicando os demais. Além disso, o monitoramento contínuo do uso de recursos permite identificar gargalos e agir proativamente. O controle adequado de recursos assegura um ambiente equilibrado e confiável para uma melhor experiência do usuário e gerenciamento eficiente dos recursos disponíveis.

Interação entre containers



A interação entre containers no Docker é facilitada pelos comandos "exec" e "cp". O comando "exec" permite executar comandos dentro de um container em execução, fornecendo acesso ao ambiente do container para manutenção, depuração ou configuração. Já o comando "cp" possibilita a cópia de arquivos entre o host e um container em execução, permitindo o compartilhamento de dados e a transferência de informações de forma simples e prática. Essas ferramentas são fundamentais para o gerenciamento e a personalização dos containers, oferecendo maior flexibilidade e controle sobre o ambiente de execução.

Backup de containers

O controle de recursos é essencial em contêineres para evitar problemas de desempenho e instabilidade. Ao limitar o uso de CPU, memória e armazenamento, é possível evitar que um contêiner monopolize todos os recursos, prejudicando os demais. Além disso, o monitoramento contínuo do uso de recursos permite identificar gargalos e agir proativamente. O controle adequado de recursos assegura um ambiente equilibrado e confiável para uma melhor experiência do usuário e gerenciamento eficiente dos recursos disponíveis.

Execução de containers

Executando um contêiner

```
docker run nome_da_imagem:tag
```

Especificando configurações adicionais

```
docker run -p porta_host:porta_container nome_da_imagem:tag
```

Interagindo com container em execução

```
docker logs <ID_DO_CONTAINER>
```

Gerenciando contêineres em execução:

```
docker ps
docker stop <ID_DO_CONTAINER>
docker rm <ID_DO_CONTAINER>
```

Removendo Imagens

Removendo uma imagem específica:

```
docker rmi nome_da_imagem:tag
```

Removendo todas as imagens não utilizadas:

```
docker image prune
```

Removendo todas as imagens, incluindo as em uso:

```
docker image prune -a
```

É importante lembrar que, ao remover uma imagem, você não poderá mais executar contêineres baseados nessa imagem, a menos que a baixe novamente.

Obter Informações (docker logs, inspect)

Obter logs de um contêiner:

```
docker logs ID_DO_CONTAINER
```

Obter informações detalhadas de um contêiner:

```
docker inspect ID_DO_CONTAINER
```

Interagindo (docker exec, cp)

Executar comandos em um contêiner em execução:

```
docker exec ID_DO_CONTAINER comando
```

Copiar arquivos para dentro ou fora de um contêiner:

```
docker cp CAMINHO_DO_ARQUIVO ID_DO_CONTAINER:DIRETORIO_DESTINO
```

Fazendo Backup (docker import/export)

Exportar um contêiner:

```
docker export ID_DO_CONTAINER > arquivo.tar
```

Importar um container:

```
docker import arquivo.tar NOME_DA_IMAGEM:TAG
```

Variáveis Suportadas em Imagens

Definir uma variável de ambiente ao executar um contêiner:

```
docker run -e NOME_DA_VARIAVEL=valor nome_da_imagem
```

Definir **<u>várias</u>** variáveis de ambiente ao executar um contêiner:

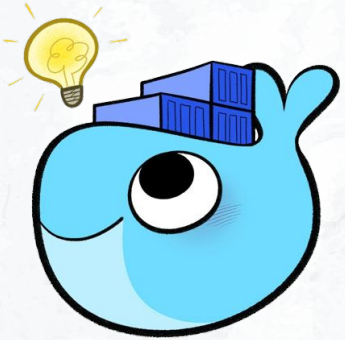
```
docker run -e VARIAVEL1=valor1 -e VARIAVEL2=valor2 nome_da_imagem
```

Usar variáveis de ambiente em um Dockerfile:

```
ENV NOME_DA_VARIAVEL valor
```

Mão na massa:

Subindo um container Mysql e criando uma base de dados.



Persistindo Dados com Volumes/Bind Mounts

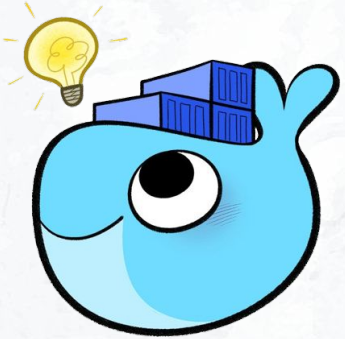
Criar um Volume:

```
docker run -v nome_do_volume:diretorio nome_da_imagem
```

Montar um diretório do host como Bind Mount:

```
docker run -v caminho_do_diretorio:diretorio nome_da_imagem
```

Mão na massa
Persistindo dados de contêineres em execução.



Acesso externo - Mapeamento de portas.

Mapear uma porta do contêiner para uma porta do host:

```
docker run -p porta_do_host:porta_do_contêiner nome_da_imagem
```

Mapear uma porta específica do host para uma porta aleatória do contêiner:

```
docker run -p porta_do_host nome_da_imagem
```

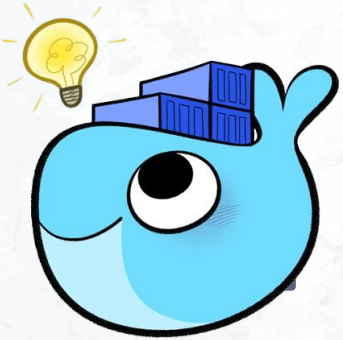
Projeto:

- Vamos subir 2 sites para acesso externo, porém, os containers serão acessados via **web proxy**.

- Não faremos mapeamento nos serviços http. Somente no proxy.

HAPROXY

APACHE




NERDS
<\UFC>