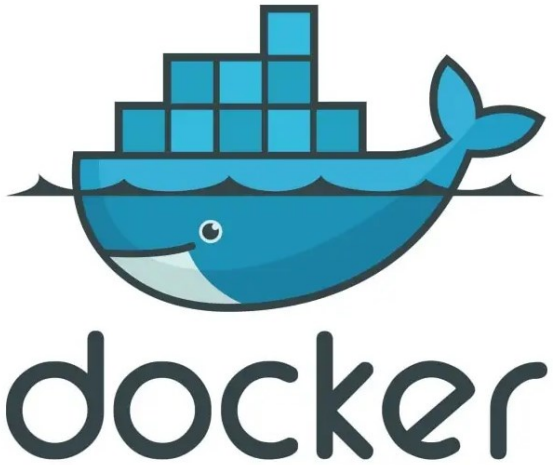


Aprendendo conceitos e praticas para proficiência

no desenvolvimento e entrega de produtos.

awtech
Technology for business

Sumário

1 Historia & Panorama

1- História e panorama

1.1 - Os Desafios do Desenvolvimento Tradicional

Antes, configurar um ambiente de desenvolvimento **exigia instalar manualmente** cada componente, como servidores, bancos de dados e dependências. **Qualquer incompatibilidade** entre sistemas ou versões **causava falhas imprevisíveis**. Além disso**, replicar o mesmo ambiente** em outra máquina **era trabalhoso e demorado**, aumentando os riscos de erros.

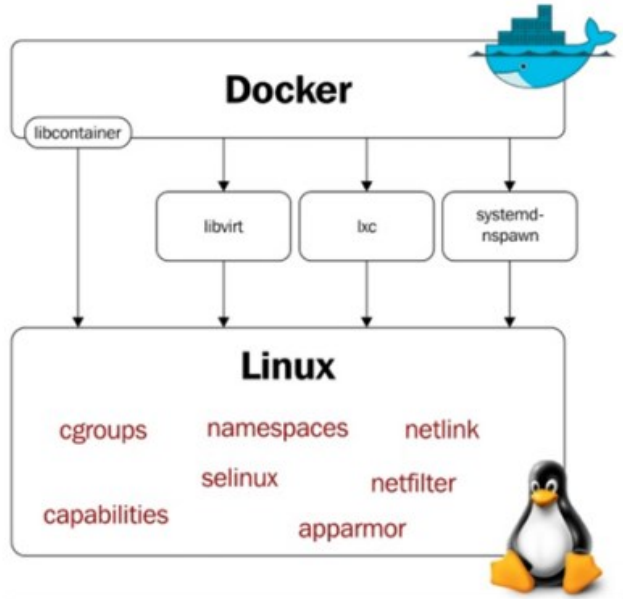
O Docker trouxe uma revolução ao desenvolvimento de software ao simplificar a configuração e a gestão de ambientes, ele permite criar **ambientes consistentes e isolados com facilidade**. Isso **elimina problemas de compatibilidade e facilita a replicação** do ambiente de desenvolvimento em diferentes sistemas, aumentando a eficiência, a agilidade e a confiabilidade nas entregas de software.

1- História e panorama

1.2 – Mas o que é o Docker ?

É uma ferramenta que se apoia em recursos existentes no kernel, inicialmente Linux, para **isolar a execução de processos**. As ferramentas que o Docker traz são basicamente uma camada de administração de containers, baseado originalmente no LXC. Podemos concluir dizendo que estes recursos já existiam no kernel a um certo tempo, **o que o Docker nos trouxe foi uma maneira simples e efetiva de utiliza-los.**

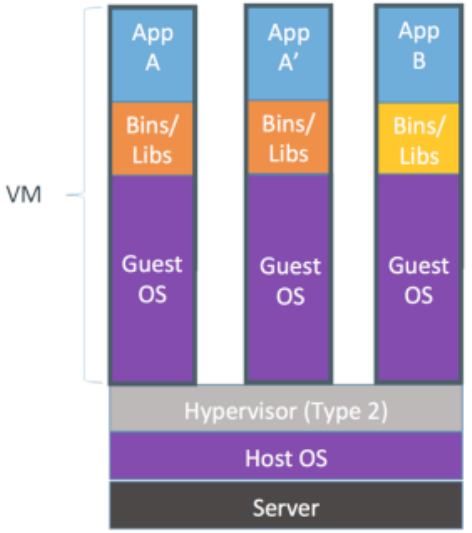
- **LXC (Linux Containers)** é uma tecnologia de virtualização leve que permite executar múltiplos sistemas Linux isolados em um único host.
- **O kernel** é o núcleo central de um sistema operacional. Ele atua como uma ponte entre o hardware do computador e os programas que rodam no sistema, gerenciando recursos como memória, processador, dispositivos de entrada/saída (disco, teclado, mouse, etc.) e a comunicação entre processos.

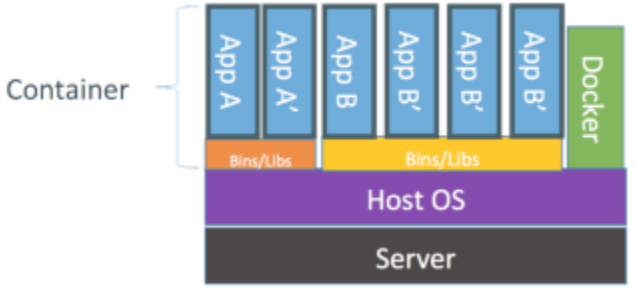


1- História e panorama

1.3 – E porque não uma marquinha virtual ?

O Docker tende a **utilizar menos recursos** que uma VM tradicional, um dos motivos é não precisar de uma pilha completa como vemos a seguir. **O Docker utiliza o mesmo kernel do host**, e ainda pode compartilhar bibliotecas. Utilizando o mesmo kernel é possível utilizar outra distribuição com versões diferentes das bibliotecas e aplicativos. Ou Seja, posso estar rodando o Daemon em um Ubuntu e ter containers rodando em cima do Fedora e outro em Mint.



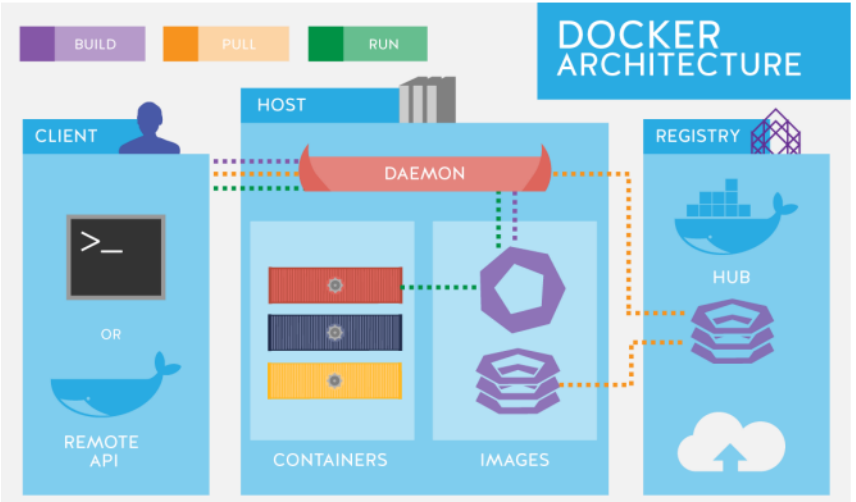


1- História e panorama

1.4 – Arquitetura

A arquitetura do **Docker Engine** é composta por três principais componentes, que trabalham em conjunto para permitir a criação, execução e gerenciamento de contêineres. Esses componentes são:

O Docker Client é a interface de linha de comando (CLI) que os usuários interagem diretamente. Ele envia comandos para o Docker Daemon, que executa as ações solicitadas, como criar contêineres, listar imagens ou manipular volumes



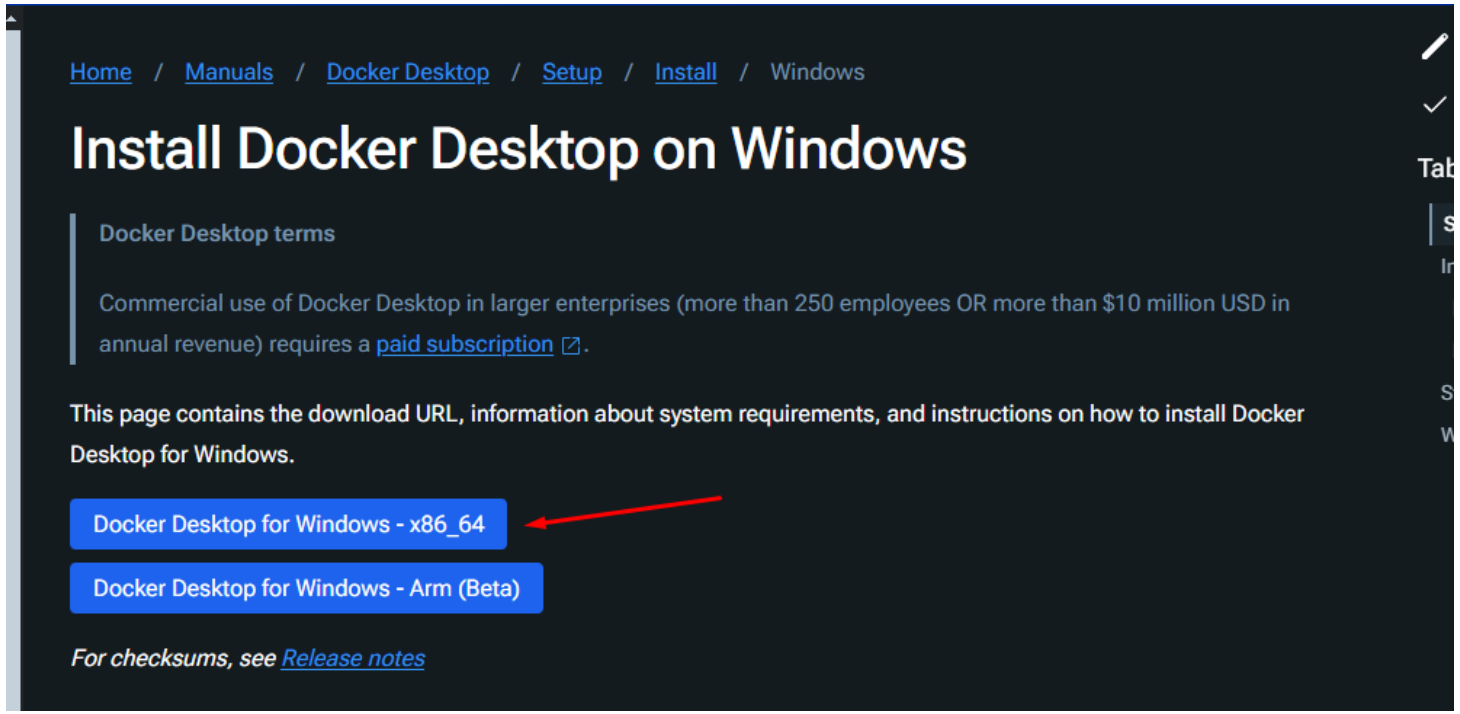
O **Docker Hub** é o registry público mais popular, mas também é possível configurar registries privados. Quando você puxa (pull) ou envia (push) imagens, está interagindo com o Docker Registry.

O Docker Daemon é o processo principal que executa no sistema e gerencia a execução dos contêineres. Ele escuta as requisições do cliente Docker e interage com o sistema operacional para construir, executar e orquestrar contêineres

1- História e panorama

1.3 – Instalação

Acessem o link e sigam os passo a passo.



https://docs.docker.com/desktop/setup/install/windows-install/

2 Conceitos Básicos

2 - Conceitos Básicos

2.1 – Imagens X Containers

Uma **imagem Docker** é um modelo imutável que contém tudo o que é necessário para rodar um aplicativo: sistema de arquivos, dependências, bibliotecas e configurações. As imagens são criadas a partir de um **Dockerfile**, que define as etapas para configurar o ambiente. Por serem compostas de camadas, as imagens são leves e compartilham partes comuns entre containers, otimizando o uso de armazenamento e transferências.

Um **container é uma instância em execução de uma imagem**. Ele encapsula o aplicativo e suas dependências em um ambiente isolado, mas leve, compartilhando o kernel do sistema operacional do host.

2 - Conceitos Básicos

2.1 – Volumes

Volumes no Docker são diretórios externos ao container que podem ser montados dentro dele. Diferente do sistema de arquivos interno do container, que segue o padrão de camadas e é volátil, os volumes são independentes e persistem os dados, mesmo que o container seja parado ou removido. A principal função dos volumes é garantir a persistência de dados. Isso significa que, enquanto o sistema de arquivos do container apaga tudo ao ser destruído, os dados armazenados em volumes permanecem intactos e acessíveis. É uma solução simples, prática e essencial para armazenar informações de maneira segura e duradoura no Docker.

1. VOLUMES ANÔNIMOS:
    1. Criados automaticamente sem nome.
    2. Usados para persistência temporária de dados.

2. VOLUMES NOMEADOS:
    1. Criados com um nome específico.
    2. Facilidade de gerenciamento e persistência duradoura.

3. BIND MOUNTS:
    1. Diretórios/arquivos específicos do host montados no container.
    2. Alterações no host são refletidas no container em tempo real.

2 - Conceitos Básicos

2.1 – Networks (Rede)

Network são usadas para conectar containers entre si e com o mundo externo, permitindo a comunicação entre eles. O Docker fornece vários tipos de redes predefinidas e permite a criação de redes personalizadas para atender a diferentes necessidades de isolamento e comunicação.

- BRIDGE:
  - Rede padrão para containers que não são conectados a outras redes.
  - Os containers conectados a essa rede podem se comunicar entre si, mas não com o host diretamente, a não ser por meio de portas expostas.
- HOST:
  - O container compartilha a rede do host, sem isolamento de rede.
  - Usado quando você precisa de uma rede semelhante à do host, por exemplo, para aplicações de alta performance.
- NONE:
  - Nenhuma rede é atribuída ao container.
  - O container não tem acesso à rede, útil para situações de isolamento completo.
- OVERLAY:
  - Usada para criar redes entre containers em diferentes hosts Docker, normalmente em ambientes de swarm.
  - Permite a comunicação entre containers distribuídos em vários nós.
- MACVLAN:
  - Permite que containers apareçam como dispositivos físicos na rede, com seus próprios endereços MAC.
  - Ideal para integração com redes externas ou quando um container precisa se comportar como um dispositivo físico de rede.

2 - Conceitos Básicos

2.1 – Composer

O Docker Compose é uma ferramenta usada para **definir e gerenciar múltiplos containers** Docker de maneira simples e organizada. Com o Docker Compose, é possível configurar serviços, redes e volumes necessários para um aplicativo completo usando um único arquivo de configuração chamado docker-compose.yml.

Em vez de executar múltiplos comandos docker run para iniciar containers de maneira individual, **você usa o Docker Compose para orquestrar a criação, execução e monitoramento desses containers em conjunto**, facilitando a gestão de aplicações que envolvem vários serviços (como bancos de dados, servidores web, etc.).

2 - Conceitos Básicos

2.1 – Composer

O Docker Compose é uma ferramenta usada para **definir e gerenciar múltiplos containers** Docker de maneira simples e organizada. Com o Docker Compose, é possível configurar serviços, redes e volumes necessários para um aplicativo completo usando um único arquivo de configuração chamado docker-compose.yml.

Em vez de executar múltiplos comandos docker run para iniciar containers de maneira individual, **você usa o Docker Compose para orquestrar a criação, execução e monitoramento desses containers em conjunto**, facilitando a gestão de aplicações que envolvem vários serviços (como bancos de dados, servidores web, etc.).

2 - Comandos

1. **Comandos de Volumes**

- `docker volume create <nome>`:
  - Cria um novo volume com o nome especificado.
- `docker volume ls`:
  - Lista todos os volumes existentes no sistema.
- `docker volume inspect <nome_do_volume>`:
  - Exibe detalhes sobre o volume, como onde ele está armazenado e quais containers o utilizam.
- `docker volume rm <nome_do_volume>`:
  - Remove o volume especificado.
- `docker volume prune`:
  - Remove volumes não utilizados (volumes que não estão associados a nenhum container).

2 - Comandos

2. **Comandos de Redes (Networks)**

- `docker network create <nome_da_rede>`:
  - Cria uma nova rede personalizada.
- `docker network ls`:
  - Lista todas as redes criadas no Docker.
- `docker network inspect <nome_da_rede>`:
  - Exibe detalhes sobre uma rede específica, incluindo os containers conectados a ela.
- `docker network connect <nome_da_rede> <nome_do_container>`:
  - Conecta um container a uma rede específica.
- `docker network disconnect <nome_da_rede> <nome_do_container>`:
  - Desconecta um container de uma rede.
- `docker network prune`:
  - Remove redes não utilizadas (redes que não estão conectadas a nenhum container).

2 - Comandos

3. Comandos de Containers

- `docker run <opções> <imagem>`:
  - Cria e executa um novo container a partir de uma imagem especificada. Exemplo: `docker run -d -p 8080:80 nginx`.
- `docker ps`:
  - Lista todos os containers em execução no momento.
- `docker ps -a`:
  - Lista todos os containers, incluindo os que estão parados.
- `docker rm <nome_ou_id_do_container>`:
  - Remove um container que foi parado (não pode estar em execução).
- `docker logs <nome_ou_id_do_container>`:
  - Exibe os logs de saída de um container.
- `docker exec -it <nome_ou_id_do_container> <comando>`:
  - Executa um comando dentro de um container em execução (por exemplo, iniciar um shell interativo).
- `docker stop <nome_ou_id_do_container>`:
  - Para a execução de um container em execução.
- `docker start <nome_ou_id_do_container>`:
  - Inicia um container que está parado.
- `docker restart <nome_ou_id_do_container>`:
  - Reinicia um container em execução.

2 - Comandos

4. Comandos de Imagens

- `docker pull <imagem>`:
  - Baixa uma imagem do Docker Hub ou de outro repositório.
- `docker images`:
  - Lista todas as imagens armazenadas localmente no sistema.
- `docker build -t <nome_da_imagem> <caminho>`:
  - Cria uma imagem a partir de um Dockerfile localizado no caminho especificado.
- `docker rmi <nome_ou_id_da_imagem>`:
  - Remove uma imagem localmente.
- `docker tag <imagem> <nome_da_imagem:tag>`:
  - Atribui uma tag a uma imagem existente, o que facilita o gerenciamento e versionamento.
- `docker push <nome_da_imagem>`:
  - Envia uma imagem local para um repositório, como o Docker Hub.

3 Além Da Teoria