



DESENVOLVIMENTO DE APIS E MICROSSERVIÇOS

Aula 01 – Docker

Caio Eduardo do Prado Ireno
caio.ireno@faculdadeimpacta.com.br

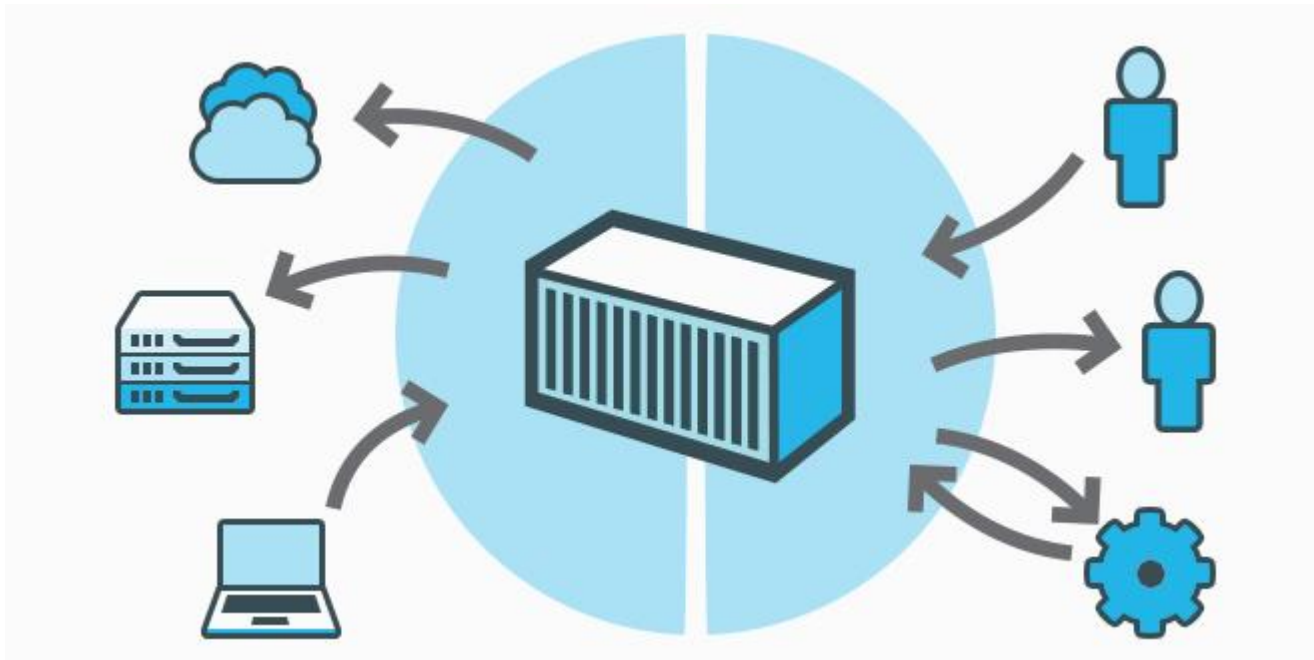
Containers

Como nossa aula é focada em Docker, não dá para falarmos sobre ele sem antes apresentar um entendimento básico e breve sobre **containers**.

Apesar de os containers não serem exatamente o mesmo que Docker, você perceberá o grande impacto que o Docker teve nesse universo.

O que são containers

"Um container é uma unidade personalizada de software que empacota o código e todas as suas dependências para que um software possa ser executado de forma rápida, consistente e em qualquer ambiente"



O que são containers

Principais características:

- Imutabilidade: Uma vez que o container é colocado em execução, ele é projetado para rodar de forma eficiente e sempre de maneira idêntica, toda vez que for iniciado.
- Isolamento de processos e recursos computacionais: Não afetem a máquina host (a máquina que está rodando nosso processo operacional) pois eles conseguem isolar processos, rede, memória, CPU e outros recursos computacionais.

O que são containers

Principais características:

- Leves: Diferentemente de máquinas virtuais, os containers são muito rápidos. Basicamente, são processos rodando no sistema operacional, podendo iniciar e encerrar rapidamente.

Além disso, um container não precisa realizar o boot como um sistema operacional tradicional.

O que são containers

Principais características:

Ele utiliza o kernel do sistema operacional host para executar seus processos. O container "acredita" ser um sistema operacional completo, mas na verdade opera com volumes e sistemas de arquivos montados que o permitem acessar os recursos do kernel sem causar impacto no host.

Dentro de um container, as aplicações têm a ilusão de estar operando em um sistema operacional completo. Essa "enganação" oferecem um ambiente isolado e controlado para cada container.

O que são containers

Principais características:

“Na minha máquina funciona”

Um dos grandes benefícios dos containers é a eliminação do famoso problema de incompatibilidade entre ambientes de desenvolvimento e produção. Com containers, o ambiente é empacotado junto à aplicação, garantindo que funcione em qualquer lugar.

O que são containers

Principais características:

Linux

Embora o Docker seja compatível com outros sistemas operacionais, ele foi originalmente projetado para Linux e aproveita tecnologias do kernel Linux para fornecer isolamento e controle de recursos.

Em resumo

- Imutabilidade
- Isolamento de processos e recursos computacionais
- Leves: è executado como um processo no SO
- Utilizam recursos do kernel do SO – Não possui a necessidade de instalar um novo SO
 - São enganados , pensam que ele tem um SO próprio
- Rápidos de iniciar e de ser removido ou parado – Não há necessidade de “boot”
- Utilizam “imagens” (imutável) para serem executados
- “na minha maquina funciona”
- Linux

Em resumo

O container compartilha o kernel do sistema operacional host, ou seja, utiliza o kernel da máquina principal.

Isso permite isolar apenas os processos que queremos rodar, tornando o processo muito mais leve, rápido e eficiente, com uma inicialização quase instantânea.

Um container funciona como um software: você clica duas vezes, ele inicia; manda fechar, ele para. É basicamente isso.

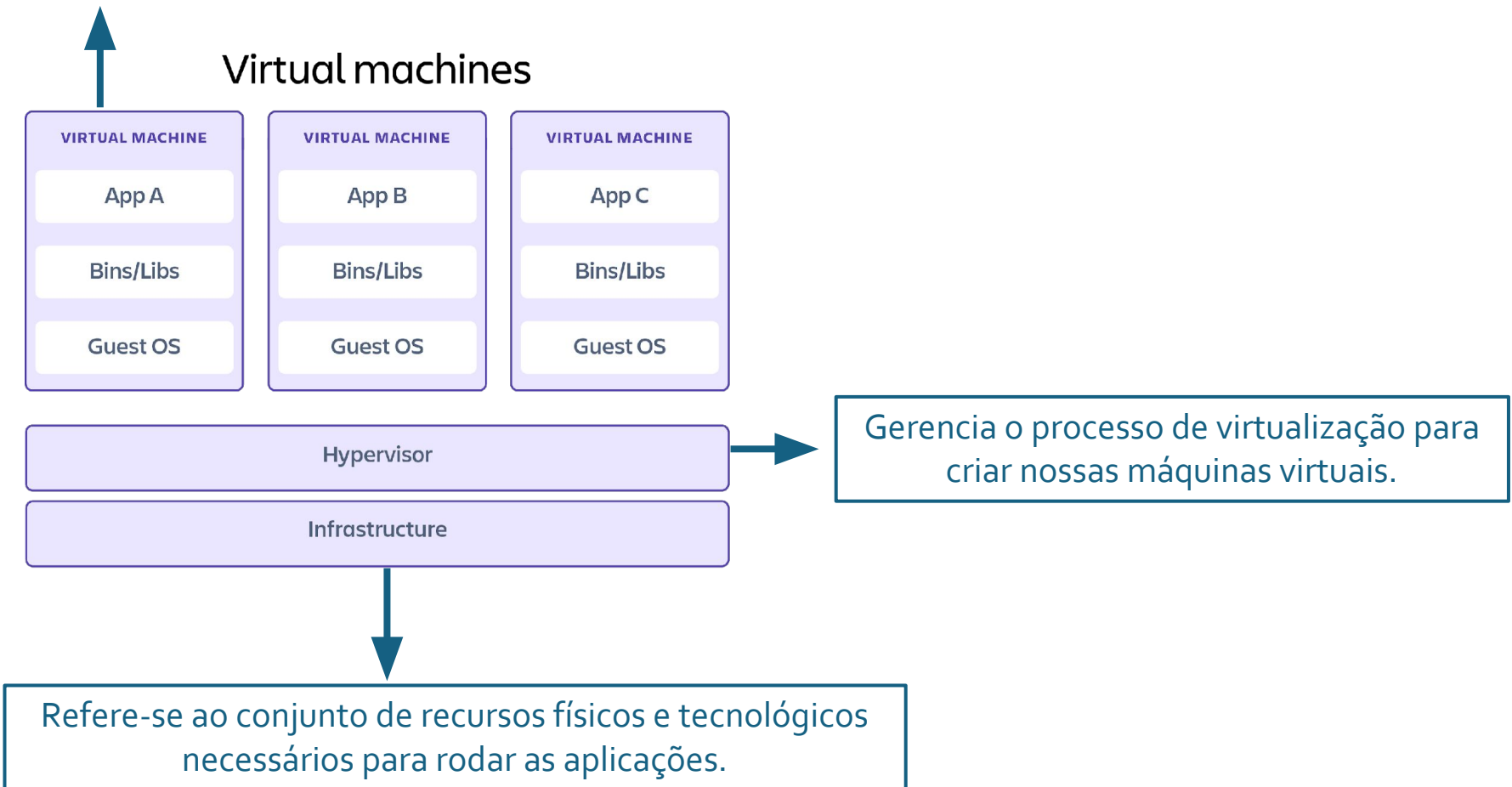
Além disso, se precisar de um software que utilize apenas 1 MB de memória, o container pode ser configurado para consumir apenas 5 MB, sem a necessidade de instalar um sistema operacional completo.

Todo recurso necessário é compartilhado com o sistema operacional host, o que gera uma economia significativa de recursos e torna o processo muito mais rápido.

E máquinas virtuais?

"Uma máquina virtual executa um sistema operacional completo, incluindo um kernel e um hypervisor que faz todo o gerenciamento de virtualização para permitir a execução de várias máquinas virtuais no mesmo hardware. Isso faz com que, de certa forma, você tenha um alto consumo de recursos, como CPU, além de um tempo maior de inicialização e uma menor eficiência no aproveitamento desses recursos."

- Aplicação (App A): O software específico que você quer executar.
- Bibliotecas e Binários (Bins/Libs): Dependências necessárias para a aplicação rodar, como bibliotecas específicas e arquivos binários.
- Sistema Operacional Convidado (Guest OS): Um sistema operacional completo, como Windows, Linux ou macOS, que roda dentro da máquina virtual

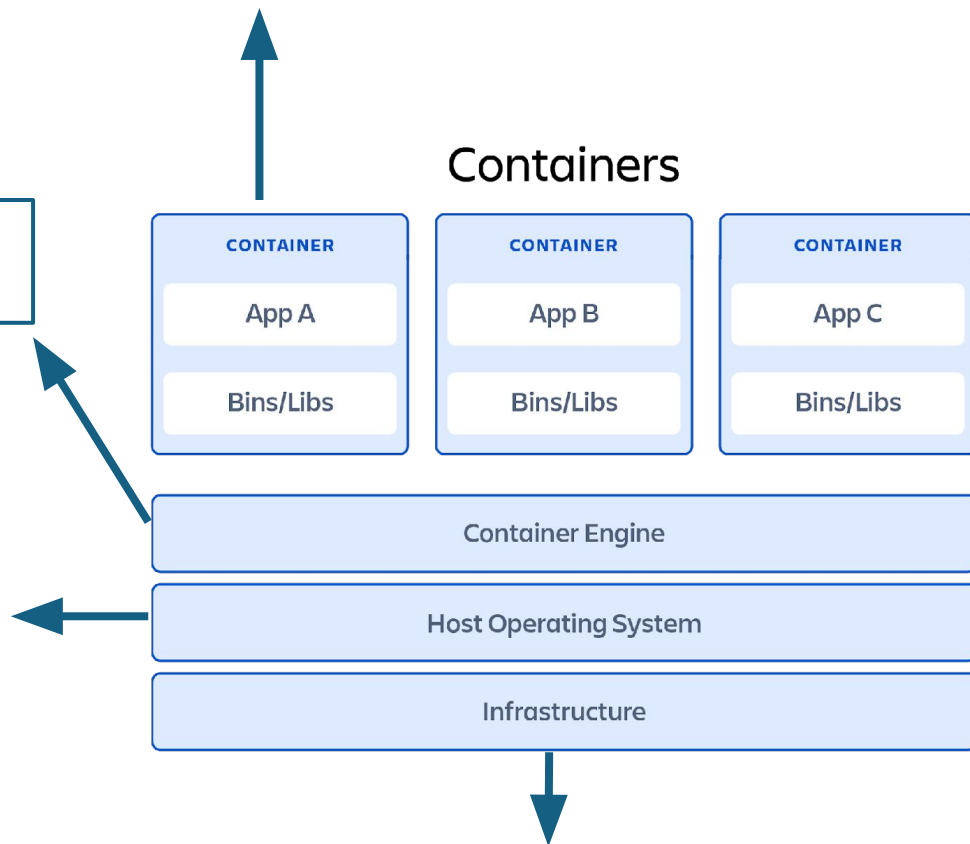


Os containers são unidades de execução criadas a partir de imagens. Cada container contém:

1. **Aplicação (App A):** O software que você deseja rodar.
2. **Binários e bibliotecas (Bins/Libs):** Dependências necessárias para a aplicação funcionar.
3. **Ambiente isolado:** A aplicação no container acredita que está rodando em um sistema próprio, graças ao isolamento fornecido pelo Docker.

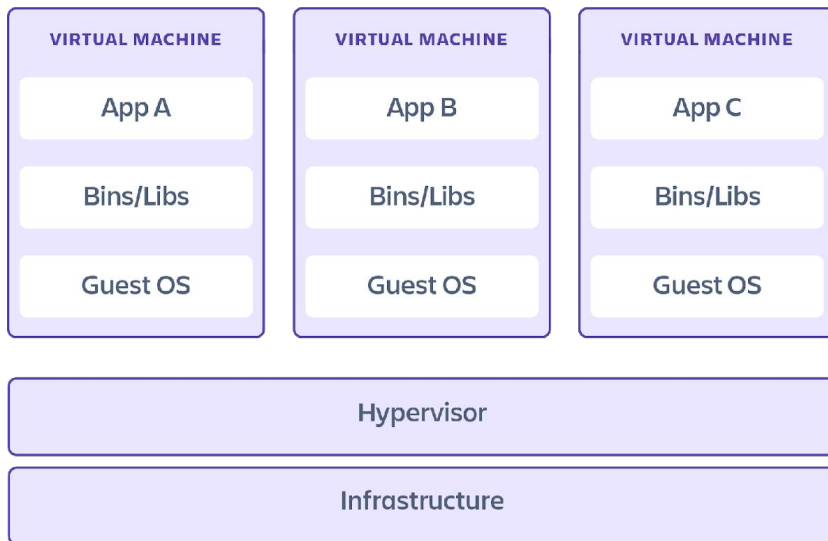
O Docker é o motor que gerencia a criação, execução e remoção de containers.

O Host OS é o sistema operacional que roda diretamente no hardware físico.
Ele fornece o **kernel** que será compartilhado por todos os containers.

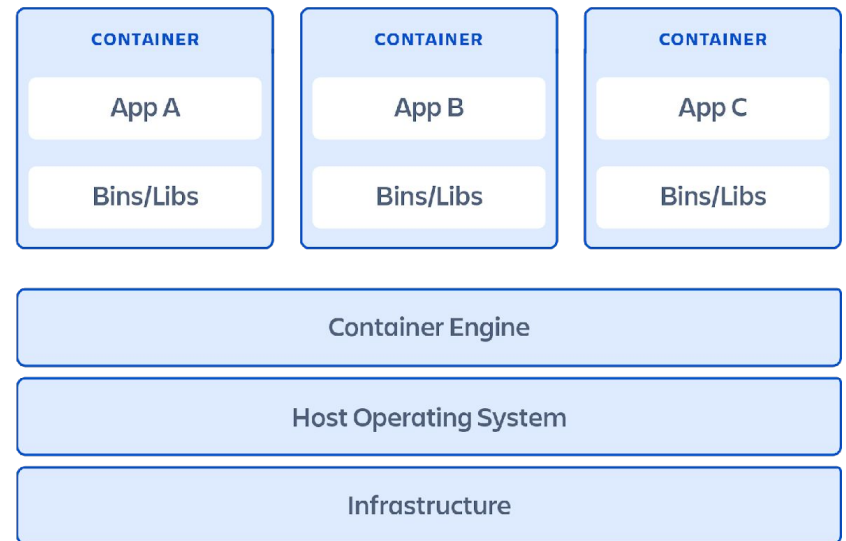


Refere-se ao conjunto de recursos físicos e tecnológicos necessários para rodar as aplicações.

Virtual machines



Containers



Container Runtimes - Docker

Container runtime é um software que permite a execução de containers. Isso significa que ele possui todas as especificações necessárias para que um container possa ser executado.

A Docker, junto com outras empresas, criou a iniciativa **chamada Open Containers Initiative**. Essa iniciativa estabeleceu uma padronização que permite que qualquer container seja executado da mesma forma, independentemente do container runtime utilizado.

Por exemplo, se você usar um container runtime diferente do Docker, como o Podman, poderá executar o mesmo container sem problemas. Essa padronização é fundamental para garantir a interoperabilidade entre os diferentes runtimes e facilitar o uso de containers em diversos ambientes.

Open Containers Initiative

A **Open Containers Initiative** pode ser acessada em opencontainers.org, onde você encontrará maiores informações sobre essa padronização.

Embora o Docker e a empresa Docker sejam um dos containers mais falados e conhecidos no mundo, é importante ressaltar que eles não foram os criadores do conceito.

Existem diversos container runtimes no mercado, porém nosso foco será especificamente no Docker.

Docker

A ideia da DotCloud era de criar uma solução de alto nível onde qualquer pessoa pudesse usar com facilidade.

O Docker se destacou por ser uma ferramenta completa, capaz de gerenciar containers, trabalhar com redes, volumes e criar builds de imagens reutilizáveis ou seja, o Docker é uma ferramenta 360 graus que gerencia o container.

Docker

O Docker opera no formato **client-server**. Quando você usa o comando `docker` no terminal, está interagindo com a interface de linha de comando, que funciona como um cliente.

Cada comando é enviado para um servidor chamado de **docker-d** que por sua vez centraliza o gerenciamento, mas também representa um ponto único de falha. Se o `docker-d` cair, todos os containers gerenciados por ele também caem. daemon

Docker

Outro aspecto relevante é que o Docker, por padrão, requer permissões de superusuário (root) para ser executado. Essa característica pode ser explorada por vulnerabilidades e representa um risco potencial.

Entretanto, o Docker também pode ser configurado para funcionar em modo rootless, onde utiliza apenas o usuário local, eliminando a dependência de privilégios elevados.

Docker

Docker Desktop: Interface gráfica para desenvolvedores gerenciarem contêineres localmente em seus computadores.

Docker Hub: Repositório central de imagens de contêineres.

Docker build cloud: Serviço para construir imagens Docker na nuvem, otimizando o processo e aliviando a carga local.

Docker Scout: Ferramenta que fornece insights sobre imagens Docker, como dependências, vulnerabilidades e boas práticas de segurança.

Docker AI, infoSiftr – Docker Hub / Docker Verified Published, **Tilt** – Gerenciamento de ambientes de desenvolvimento para aplicações / microsserviços que rodam em Kubernetes, **AtomicJar** – TesteContainers, **Mutagen.io** synchronizes file shares

Instalação

Lembrando que o Docker foi projetado para **Linux**.

Se você utiliza Windows ou Mac, o Docker não funciona nativamente no seu computador.

Isso significa que, para rodar o Docker no seu computador, você precisa de uma máquina virtual Linux.

WSL2 + Docker

Hoje, a melhor maneira de usar o Docker no Windows é utilizando o **WSL2** (Windows Subsystem for Linux), uma solução que roda o Linux como um subsistema no Windows.

Na prática, ele é virtualizado, permitindo acesso ao Linux por meio de um terminal para rodar comandos, incluindo os do Docker.

Guia/Tutorial rápido do WSL2 + Docker

[wsl2-docker-quickstart/README.md at main · codeedu/wsl2-docker-quickstart](https://codeedu.com.br/wsl2-docker-quickstart/README.md)

WSL2 + Docker

Guia/Tutorial rápido do WSL2 + Docker

[wsl2-docker-quickstart/README.md at main · codeedu/wsl2-docker-quickstart](#)

Para o Docker recomendo

[wsl2-docker-quickstart/README.md at main · codeedu/wsl2-docker-quickstart](#)

2 - Instalar o Docker com Docker Engine (Docker Nativo)

A instalação do Docker no WSL 2 é idêntica a instalação do Docker em sua própria distribuição Linux, portanto se você tem o Ubuntu é igual ao Ubuntu, se é Fedora é igual ao Fedora. A documentação de instalação do Docker no Linux por distribuição está [aqui](#), mas vamos ver como instalar no Ubuntu.

Quem está migrando de Docker Desktop para Docker Engine, temos duas opções

1. Desinstalar o Docker Desktop.
2. Desativar o Docker Desktop Service nos serviços do Windows. Esta opção permite que você utilize o Docker Desktop, se necessário,

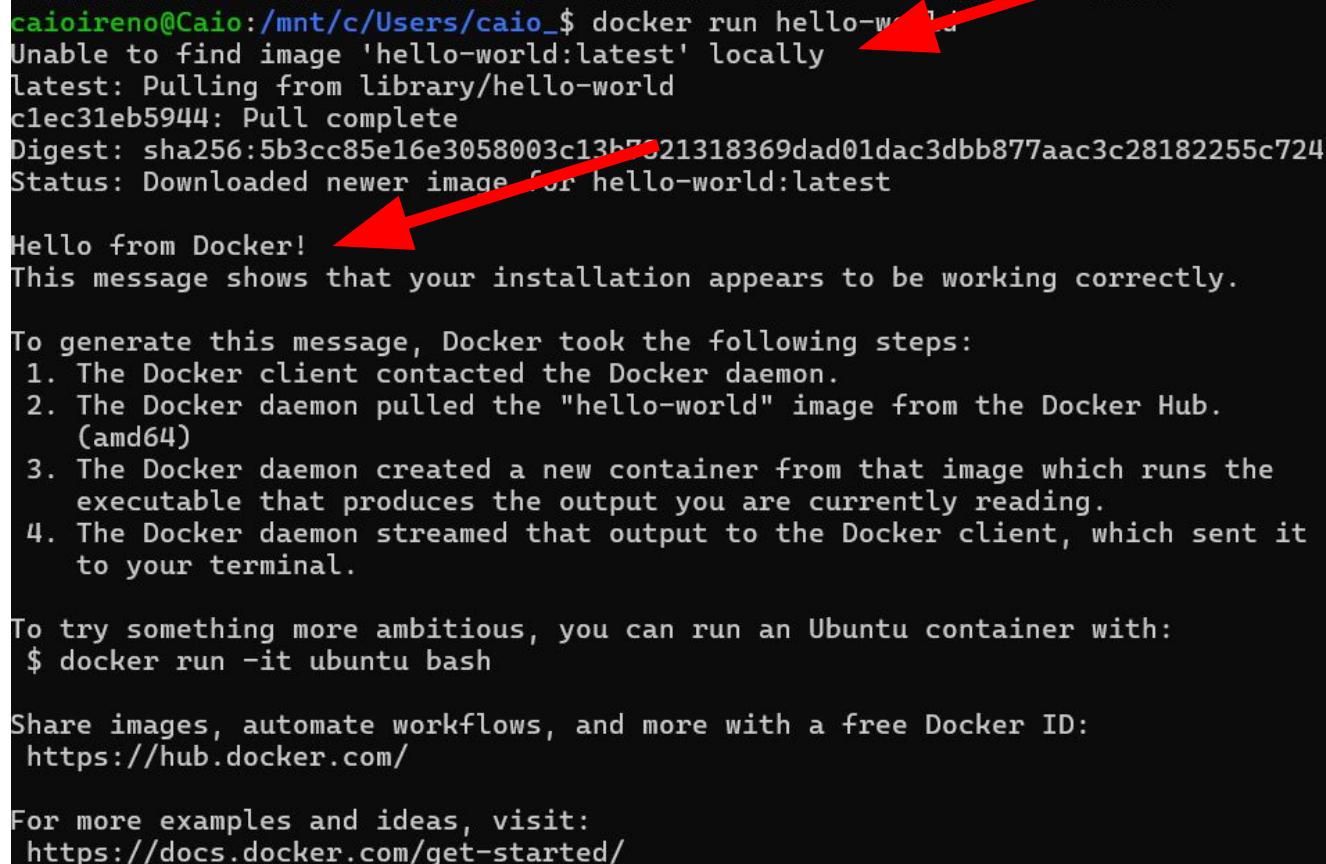
WSL2 + Docker

O tutorial de instalação possui muitas teorias e boas práticas de desenvolvimento/infraestrutura, recomendo fortemente que seja feito a leitura.

Organizem as ideias, realizem os procedimentos, anotem e entendam.

Manipulando Containers

Para verificar se o Docker está instalado corretamente, execute: `docker run hello-world`



```
caioireno@Caio:/mnt/c/Users/caio_$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:5b3cc85e16e3058003c13b7321318369dad01dac3dbb877aac3c28182255c724
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Manipulando Containers

Executando um Container com um Nome Personalizado

Por padrão, o Docker atribui nomes aleatórios aos containers. Você pode especificar um nome usando a flag **--name**.

```
docker run --name mynginx nginx
```

Manipulando Containers

Executando um Container em Segundo Plano (d)

Para executar um container em modo "detached" (segundo plano), use a flag -d

```
docker run -d --name mynginx nginx
```

Manipulando Containers

Mapeando Portas com a Flag p

Para mapear a porta do container para a porta do host, use -p

```
docker run -d -p 8080:80 nginx
```

Isso mapeia a porta 80 do container para a porta 8080 do host.

Manipulando Containers

Listando Containers em Execução e Parados

Containers em execução: `docker ps`

Todos os containers (incluindo parados): `docker ps -a`

*Diferença: **docker ps** lista apenas os containers em execução, enquanto **docker ps -a** lista todos os containers existentes no sistema.*

Manipulando Containers

Parando um Container: `docker stop mynginx`

Iniciando um Container Parado: `docker start mynginx`

Removendo um Container

Remoção normal: `docker rm mynginx`

Remoção forçada (para containers em execução): `docker rm -f mynginx`

Diferença: `docker rm` remove apenas containers parados. Para remover um container em execução, use `docker rm -f`.

Manipulando Containers

Para verificar se o Docker está instalado corretamente,
execute: `docker run hello-world`

Por que usar Docker?

Portabilidade

Com Docker, sua aplicação roda da mesma forma em qualquer lugar (seja no seu PC, no Render, AWS, ou outro servidor).

Não importa qual sistema operacional você usa ou qual versão do Python está instalada.

Por que usar Docker?

Ambiente isolado

O Docker garante que todas as dependências (Python, Flask, pacotes, banco de dados) estão **sempre iguais**, evitando o famoso "na minha máquina funciona".

Por que usar Docker?

Ambiente isolado

O Docker garante que todas as dependências (Python, Flask, pacotes, banco de dados) estão **sempre iguais**, evitando o famoso "na minha máquina funciona".

Manipulando Containers

Para FLASK

dockerfile

```
FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r
requirements.txt

COPY . .

EXPOSE 5000
```

docker build -t <nome> .

docker run -p 5000:5000 <nome>

Exemplo:

[API-s-Python/FlaskWithDocker at master · caio-ireno/API-s-Python](#)

Links e Referência

Docker Full Cycle: <https://fullcycle.com.br/categoria/docker/>

Containers vs. virtual machines:

<https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms/>

Containers are not VMs: <https://www.docker.com/blog/containers-are-not-vms/>

Docker Desktop: <https://www.docker.com/products/docker-desktop/>

Docker Hub: <https://www.docker.com/products/docker-hub/>

Docker docs: <https://docs.docker.com/>

Guia/Tutorial rápido do WSL2 + Docker:

<https://github.com/codeedu/wsl2-docker-quickstart>



Obrigado!
