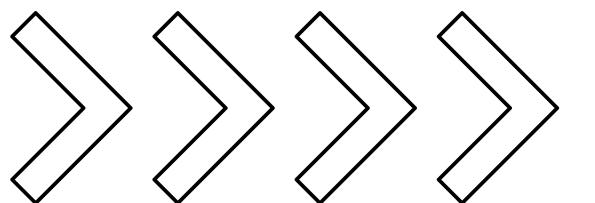


# WORKSHOP: DEVOPS

## CONTEINERIZAÇÃO COM DOCKER



Leandro Rabelo  
Rafael Moreira

# Sumário.

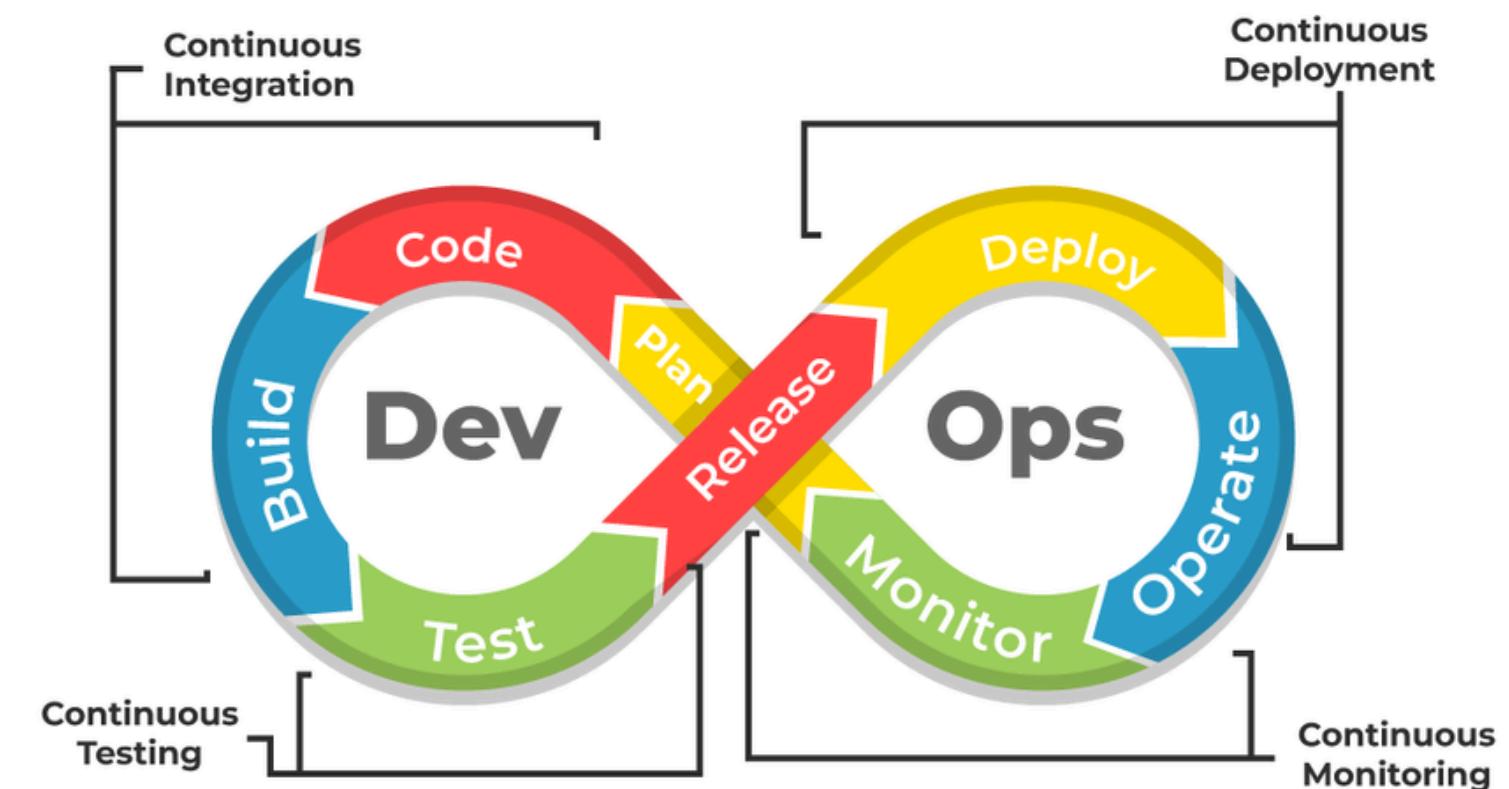
- 01. O que é DevOps ?
- 02. O que é Docker ?
- 03. Contêineres vs. Máquinas virtuais.
- 04. Componentes do Docker.
- 05. Comandos essenciais.
- 06. Instalação.
- 07. Demonstração prática.
- 08. Exercício proposto: Volumes.

# O que é DevOps ?

É um conjunto de **filosofias, práticas e ferramentas** que une as equipes de Desenvolvimento de Software (Dev) e Operações de TI (Ops).

**Objetivo principal:** Entregar aplicações e serviços de forma mais rápida, confiável e em escala.

Entre seus benefícios podemos mencionar sua **Velocidade, Entrega contínua, Confiabilidade, Escalabilidade, Colaboração** e também a **Segurança**.



**Fonte:** [geeksforgeeks.org](http://geeksforgeeks.org).

# O que é Docker ?

Docker é uma plataforma de containerização que empacota e executa aplicações em ambientes isolados e padronizados chamados contêineres.

**Isolamento:** Cada contêiner tem seu próprio código, bibliotecas e dependências.

**Portabilidade:** “Funciona na minha máquina”, agora funciona em qualquer lugar.

**Leveza:** Contêineres são muito mais leves e rápidos que Máquinas Virtuais.

**Colaboração:** Facilita criar, testar, corrigir e compartilhar ambientes completos.

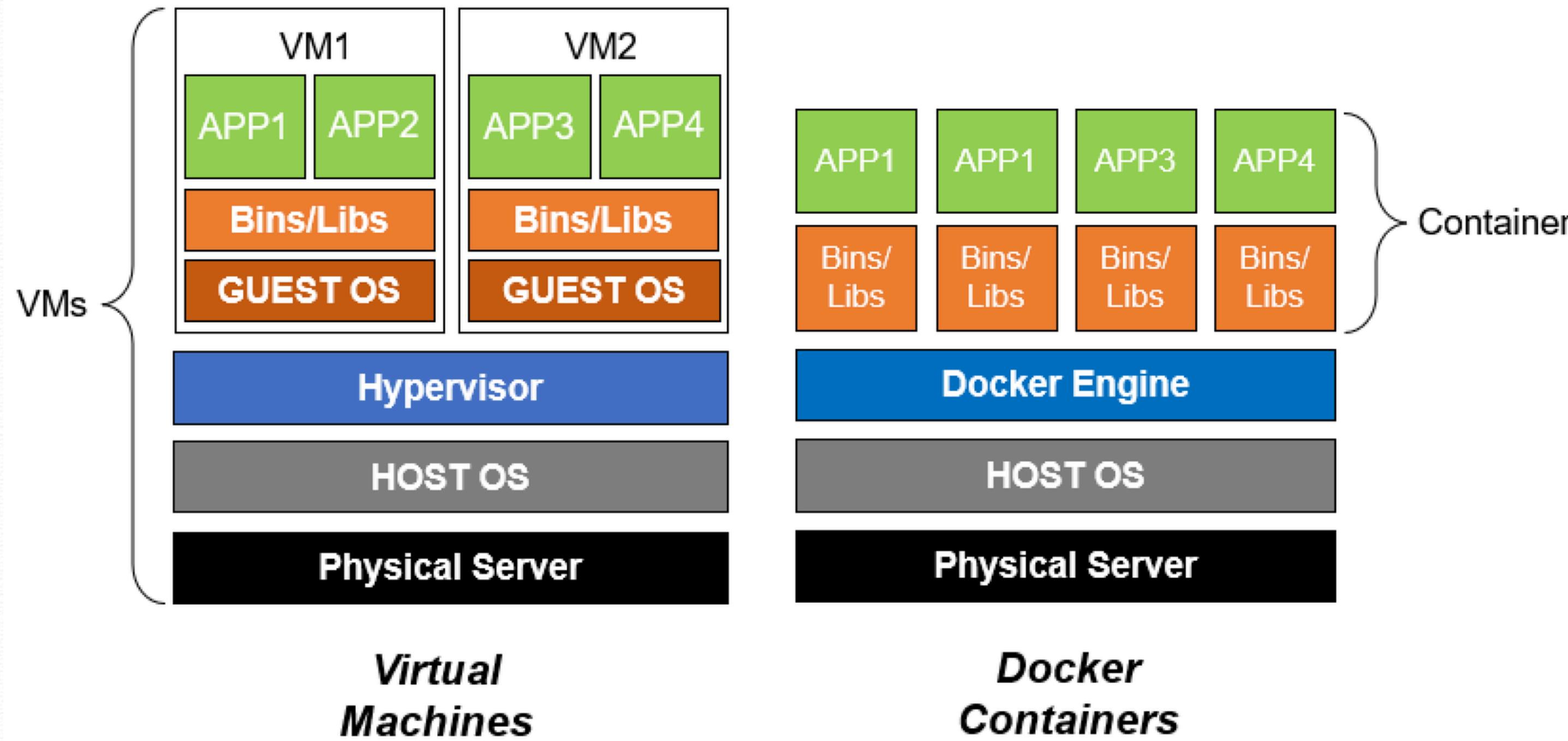


**Fonte:** wikipedia.

# Contêineres vs. Máquinas Virtuais

Característica	Contêineres (Docker)	Máquinas Virtuais (VMs)
Arquitetura	Compartilham o Kernel do SO Host	Possuem um SO Convidado completo (Guest OS)
Inicialização	Segundos	Minutos
Recursos	Baixo consumo de CPU e memória	Alto consumo de CPU e memória
Isolamento	Isolamento de processos	Isolamento completo de SO
Tamanho	Leves (Megabytes)	Pesadas (Gigabytes)
Caso de Uso Ideal	Microservices, CI/CD, Aplicações Web, Escalabilidade	Isolar sistemas operacionais diferentes, VDI, legado
Exemplos	Nginx, PostgreSQL, Aplicações Node.js	Rodar Windows em um Mac, Servidor Windows Server

# Comparação da arquitetura



Fonte: [mdpi.com](http://mdpi.com).

# Componentes

**Imagen:** Um modelo (template) somente leitura que contém instruções para criar um contêiner. É a “planta” da sua aplicação.

**Contêiner:** Uma instância em execução de uma imagem. É leve, portátil e executa a sua aplicação de forma isolada.

**Dockerfile:** Um arquivo de texto com a “receita” para construir uma imagem Docker, passo a passo.

**Docker Engine:** O motor principal do Docker. É o serviço (daemon) que cria e gerencia os contêineres.

**Docker Hub:** Um repositório público (como um GitHub de imagens) para armazenar, baixar e compartilhar imagens.

# Fluxo básico

## Imagen customizada

1. **Dockerfile**: Escrevendo a “receita” para criação da imagem.
2. **Build**: Construindo a imagem a partir da receita escrita.
3. **Run**: Executando um contêiner a partir de uma imagem.

**Imagen prontas (vindas do Docker Hub), necessitam apenas de download e execução.**

# Comandos básicos

Obs.: Para qualquer comando temos a opção **-h**, para ajuda referente ao comando, e.g. **docker ps -h**.

**docker run [imagem]** - Cria e executa um contêiner a partir de uma imagem

**docker build -t [nome-da-imagem] .** - Constrói uma imagem a partir de um Dockerfile no diretório atual.

**docker pull [imagem]** - Baixa uma imagem do Docker Hub

**docker ps** - Lista os contêineres que estão em execução. (Use **-a** para ver todos).

**docker images** - Lista todas as imagens baixadas localmente.

**doccker stop [id-do-conteiner]** - Para um contêiner em execução

**docker rm [id-do-conteiner]** - Remove um contêiner que já foi parado.

# Sintaxe básica - Dockerfile

## # INSTRUÇÃO argumentos

**FROM [imagem]** - Declara imagem base para o build.

**WORKDIR [caminho]** - Define o diretório de trabalho (Utilizado para comandos **RUN**, **CMD**, **COPY**, **ENTRYPOINT** e **ADD**).

**COPY [origem] [destino]** - Copia arquivos e diretórios do sistema host para a imagem.

**RUN [comando]** - Executa comandos úteis para a construção da imagem.

**EXPOSE [porta]** - Especifica a porta que o container irá esperar.

**CMD [executável] [parâmetro 1] [parâmetro 2]** - Especifica o comando que será usado quando o contêiner for inicializado.

# Instalação

A forma mais simples de começar é com o **Docker Desktop**.

1. Acesse [www.docker.com](http://www.docker.com).
2. Baixe o instalador para o seu sistema operacional (Windows, macOS ou Linux).
3. Siga as instruções de instalação.

## Verificando a instalação

Abra o terminal e execute os comando **docker --version**, que deve retornar a versão do seu docker, caso haja erro, a instalação foi mal sucedida.

# • • • • • • • •

## Estudo de Caso

### “Na minha máquina funciona”

**Cenário:** É requisitado a criação de uma nova funcionalidade na sua aplicação, onde com um endpoint específico, será possível acessar as configurações do usuário. Mas o desenvolvedor que irá testar essa funcionalidade tem uma versão do python em seu computador que não é a mesma ao python usado pelo servidor de produção. Bom o desenvolvedor conseguiu construir a funcionalidade, que funciona perfeitamente em seu computador, mas que no servidor ela não funciona como deveria.

#### **Conflito:**

- **Ambiente do Desenvolvedor:** Python 3.11
- **Ambiente de Produção:** Python 3.8

• • • • • • • •

• • • • • • • •

## Extra - Atividade

# Volume

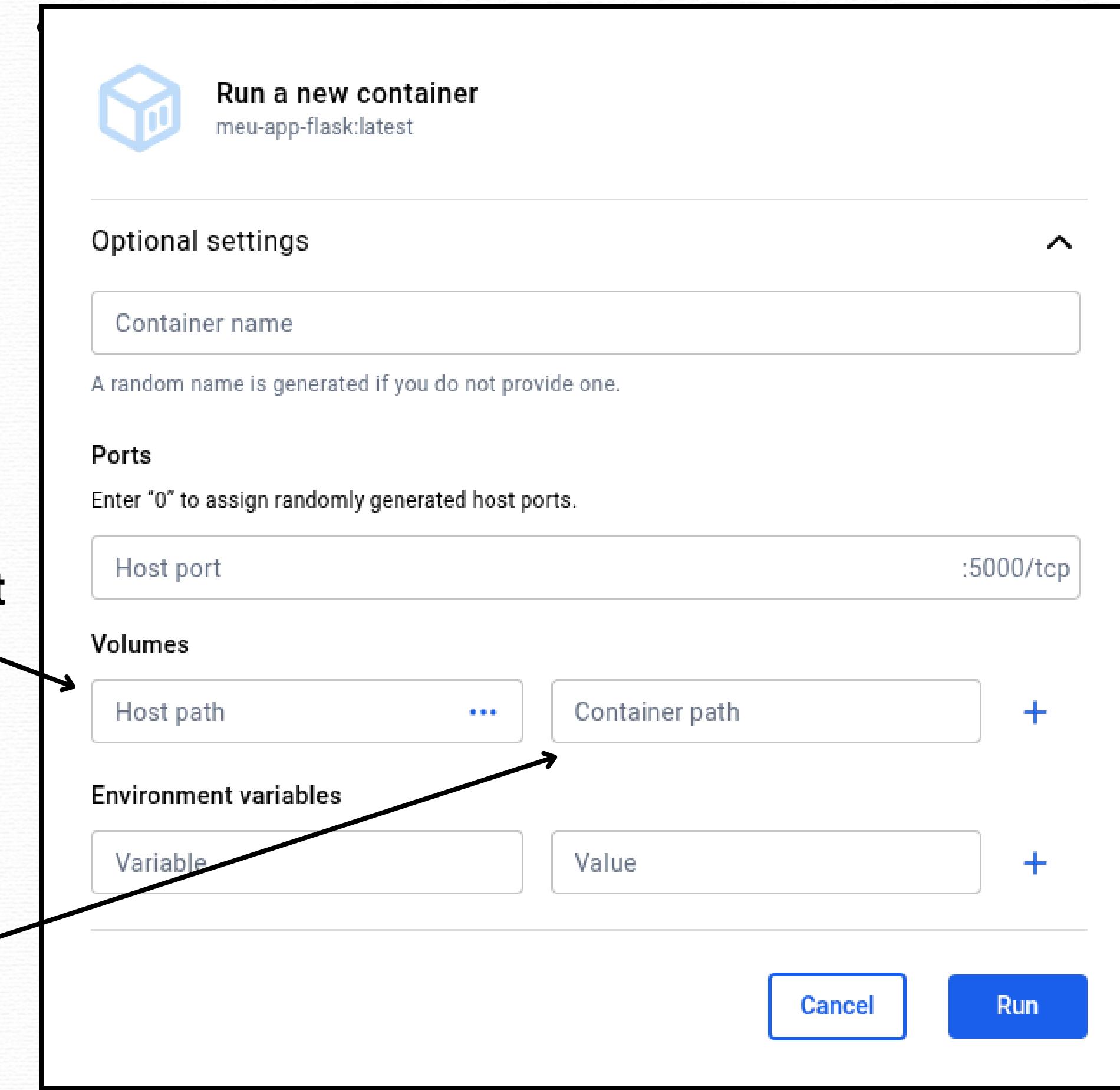
**Definição:** Volume é uma maneira de persistência nos dados dos nossos contêineres, toda vez que executamos uma imagem, um contêiner é criado, e com isso sua estrutura também é definida, onde existem os seus dados, necessários para sua execução, como uma estrutura de arquivos de um sistema operacional. Então podemos pensar que volume é apenas uma pasta compartilhada, que o contêiner a usa, sem de fato se importar onde essa pasta vai estar no SO host.

**Guia rápido:** Ao executar uma imagem, no momento de criação do contêiner, no aplicativo Docker Desktop, é possível configurar o volume nas configurações opcionais. O caminho do host, é a pasta que de fato existe no computador, onde os dados serão guardados, e o caminho do contêiner é o espelho dessa pasta, mas dentro do contêiner, como se fosse um ponteiro.

# Volume

Pasta do SO host

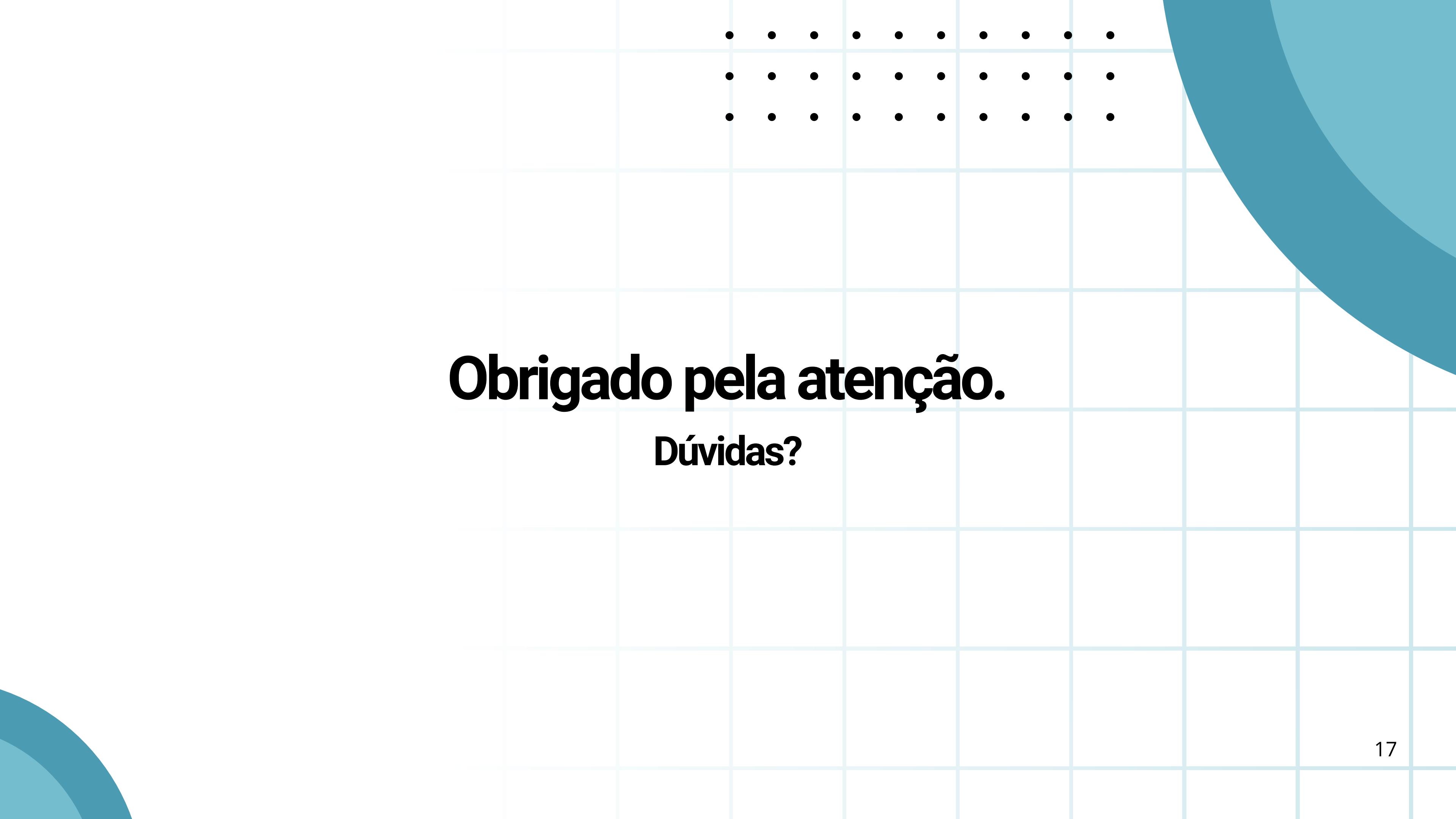
Pasta dentro do  
contêiner



# Atividade Proposta

**Descrição:** O código do projeto atual consegue salvar com sucesso os logs do sistema, mas esses logs são vistos apenas dentro dos arquivos do contêiner, existe uma pasta na raíz do projeto para guardar os logs, nomeada **logs**, quando for criar o contêiner, configure o volume para essa pasta refletir na pasta descrita no código, como **/app/logs**, ou seja, essa é a configuração apropriada para o caminho no contêiner. Após a execução do contêiner acesse o endpoint **http://localhost:5000/config**, e verifique se o sistema criou o arquivo dos logs, e o que ele escreveu nele.

**Entrega:** Para fácil entrega, crie um repositório git, com todos os arquivos do projeto, inclusive os arquivos que o sistema irá gerar, no caso, os logs. A atividade propõe apenas a configuração do volume, ou seja, não necessita de nenhuma alteração do código, mas ele pode vir a ser importante para entender o propósito da atividade.



**Obrigado pela atenção.**

**Dúvidas?**

# Referência

<https://aws.amazon.com/pt/devops/what-is-devops>

<https://www.docker.com/blog/docker-for-devops/>

<https://docs.docker.com/reference/cli/docker/>

<https://docs.docker.com/reference/dockerfile/>