Treinamento
Docker

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32

Bruno Masuda
Instrutor

Formado em MBA Gerenciamento e Projetos de Data Centers pela Faculdade Impacta Tecnologia.
Admirador e praticante da cultura DevOps/SRE com quatro anos de experiência na área.
Especializado em Kubernetes e sustentação de micro serviços.

LUIS DEV

Ementa do treinamento

O curso aborda tópicos essenciais do mundo Docker com o objetivo de demonstrar conceitos básicos sobre o funcionamento da tecnologia, visando técnicas e boas práticas para agilizar o desenvolvimento de aplicações em containers.

- Conceitos básicos Docker
- Trabalhando com Dockerfiles e distribuição de imagens (registry)
- Principais comandos
- Persistência de dados
- Docker Compose (aplicações em multi containers)

LUIS DEV

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32

Introdução Docker

LUIS DEV

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32

O que é Docker?

O Docker ajuda a abstrair tarefas de configuração repetitivas, tornando o ciclo de desenvolvimento mais ágil e facilitando a portabilidade da aplicação entre ambientes distintos, seja em um desktop, nuvem ou data center.
O ciclo de vida:

- BUILD: É onde tudo começa, a construção de uma imagem para ser executada em algum ambiente.
- SHARE: Compartilhe a imagem gerada da sua aplicação
- RUN: Execute a aplicação em qualquer ambiente Docker, seja em Desktop, em Nuvem ou servidor em data center.

LUIS DEV

O que é um container?

É um processo isolado dentro de um host.

LUIS DEV

Imagem e container

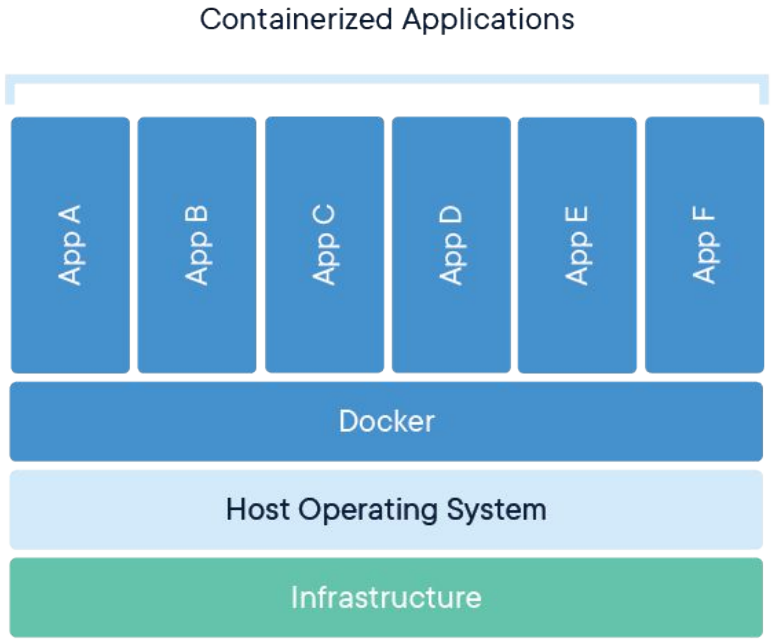
Importante destacar a diferença entre imagem docker e container docker.

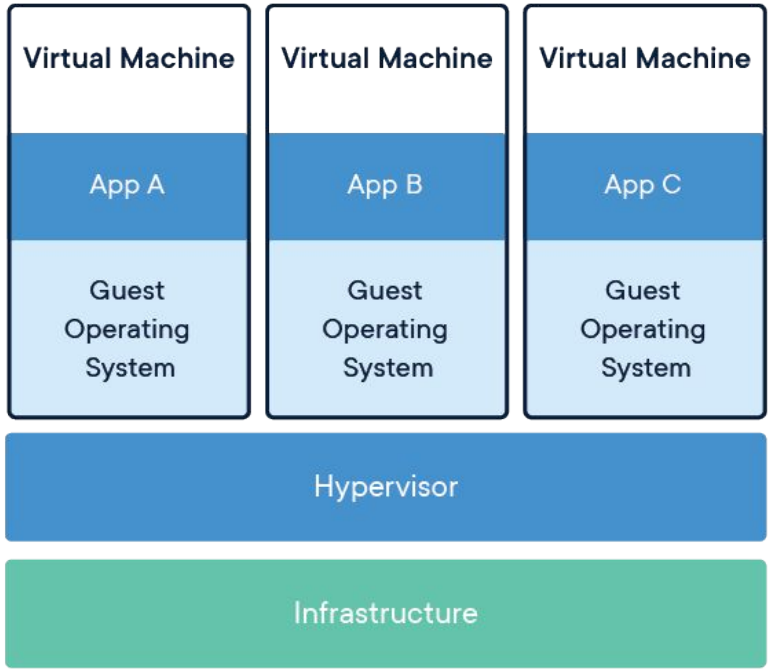
- Imagem: É o resultado de uma compilação de um Dockerfile, e é **utilizada para executar containers dentro de um host**. A imagem também é uma forma de “publicar” a aplicação em container.

- Container: **É um processo executado dentro do host** que utiliza uma imagem como base.

LUIS DEV

Diferenças entre Container e VM





LUIS DEV

Dockerfiles e registries

LUIS DEV

Dockerfiles

Dockerfiles são arquivos que contém instruções para a criação de uma imagem Docker.
Contudo, essas instruções possuem sintaxes utilizadas para a compilação (build), sendo algumas delas “RUN”, “FROM”, “CMD”, “WORKDIR” entre outras. Todas elas podem ser consultadas na documentação oficial com o detalhamento do funcionamento de cada instrução.

Para criar imagens Docker, usamos o seguinte comando:

```
$ docker build -f Dockerfile . -t registry/imagem:versão
```

LUIS DEV

Registries

Assim como nas linguagens de programação, registries docker são responsáveis por armazenar imagens, podendo compartilhá-las de forma pública ou privada entre usuários.

Alguns exemplos de registry docker:

- Docker oficial
- Azure Container Registry
- Jfrog
- Google Cloud Registry
- Amazon Elastic Container Registry
- Github Container Registry

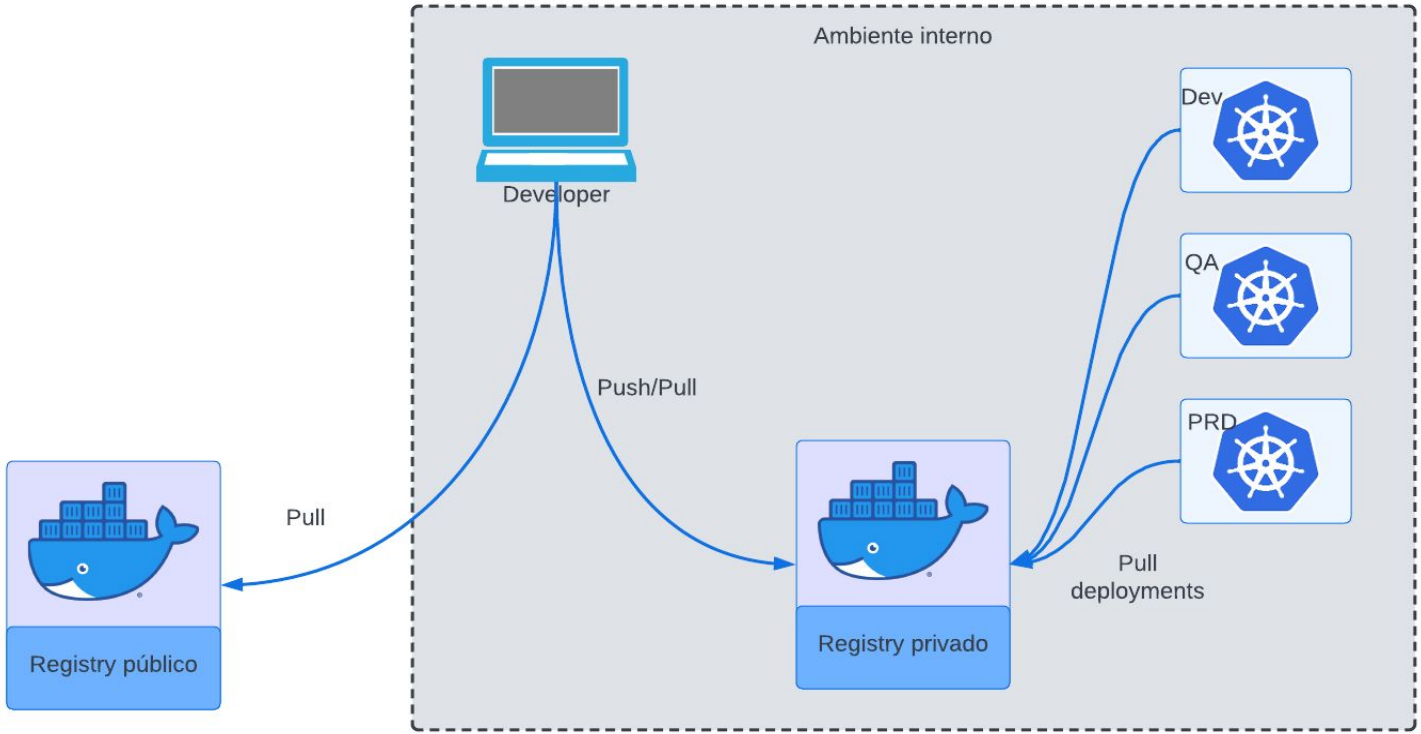
Pull e Push

Os comandos pull e push funcionam de forma semelhante ao git.

- Pull: Fazer download da imagem de um registry local/privado para o ambiente local
- Push: Fazer o upload da imagem local para um registry local/privado

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32

Registry privado x público



LUIS DEV

Exemplo Dockerfile .NET

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /App

# Copy everything
COPY . ./
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /App
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

LUIS DEV

Volumes Docker

LUIS DEV

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32

Persistência de dados em container

Por padrão, quando um container é executado através de uma imagem, todos os arquivos do sistema são originários da imagem que foi usada em sua execução.
Quando um arquivo ou dado é criado dentro de um container, esse dado/arquivo é efêmero, ou seja, não possui persistência. Caso uma aplicação em container crie um arquivo durante sua execução e depois esse mesmo container seja deletado, o arquivo gerado também é removido com o container.

Tipos de volumes

Apesar do conceito de imagens e dados efêmeros num container, existem aplicações que necessitam manter os dados, mesmo que depois o container seja deletado e/ou recriado no ambiente.

Para esses cenários, o docker disponibiliza dois tipos de montagem de volumes para manter o estado dos dados/arquivos que deverão persistir.

- Named volumes
- Bind mounts

Named volumes

São volumes gerenciados **automaticamente** e que ficam armazenados no host Docker.

Algumas vantagens são:

- Possibilidade de ser gerenciado através da linha de comando Docker
- Facilidade para criar *backups*
- Pode ser compartilhado entre vários containers

Aplicações multi containers e docker compose

LUIS DEV

O que é multi containers?

Aplicações que são compostas por mais de um container.
Por exemplo aplicações que possuem um container de *front-end* e um de *back-end* ou aplicações que precisam de um banco de dados para armazenamento de informações e entre outros.
Por boas práticas, cada container deveria executar apenas um processo:

- Melhor para controle de versão.
- Escalabilidade
- Complexidade

docker compose

Docker compose é uma ferramenta para executar e criar manifestos que executam aplicações multi containers.

Uma grande vantagem é poder definir configurações docker tais como rede, volumes, variáveis de ambientes, imagens e serviços de uma forma simples e objetiva através de manifestos "yaml".

Esses manifestos possuem uma sintaxe declarativa de fácil entendimento para quem já domina recursos docker já citados.

Francisco Rafael - francisco.rafaell.pereira@gmail.com - CPF: 068.778.653-32