



SÃO
PAULO
TECH
SCHOOL

Computação e sistemas distribuídos

Arquitetura e Containers

Eduardo Verri, Diego Brito

eduardo.verri@sptech.school

diego.brito@sptech.school

Arquitetura Monolítica – SOA – Microserviços

Arquitetura Monolítica

As principais linguagens de desenvolvimento de aplicações oferecem abstrações para quebrar a complexidade dos sistemas em módulos. Entretanto, são projetadas para a criação de um único executável monolítico, no qual toda a modularização utilizada é executada em uma mesma máquina. Assim, os módulos compartilham recursos de processamento, memória, bancos de dados e arquivos.

De modo simples podemos dizer que uma Arquitetura Monolítica é semelhante a um grande recipiente, onde todos os componentes de software de uma aplicação são empacotados e implementados juntos



Arquitetura Monolítica – desafios

Ao longo do tempo o sistema vai crescendo, se tornando mais complexo e consumindo cada vez mais recursos, o que acaba gerando alguns desafios:

Aumento da complexidade ao longo do tempo

Alta dependência de componentes de código

Escalabilidade do sistema é limitada

Falta de flexibilidade

Dificuldade para colocar alterações em produção



Arquitetura SOA



Service Oriented Architecture – composta de elementos fracamente acoplados que possuem contextos limitados.

Serviços que se comunicam entre si através de uma rede. É possível atualizar os serviços de forma independente, a atualização de um serviço não requer alteração de outros serviços

Serviços são reutilizáveis

Compartilham um contrato formal

Possuem baixo acoplamento

Abstraem a lógica

São capazes de se compor

São autônomos

Evitam alocação prolongada de recursos

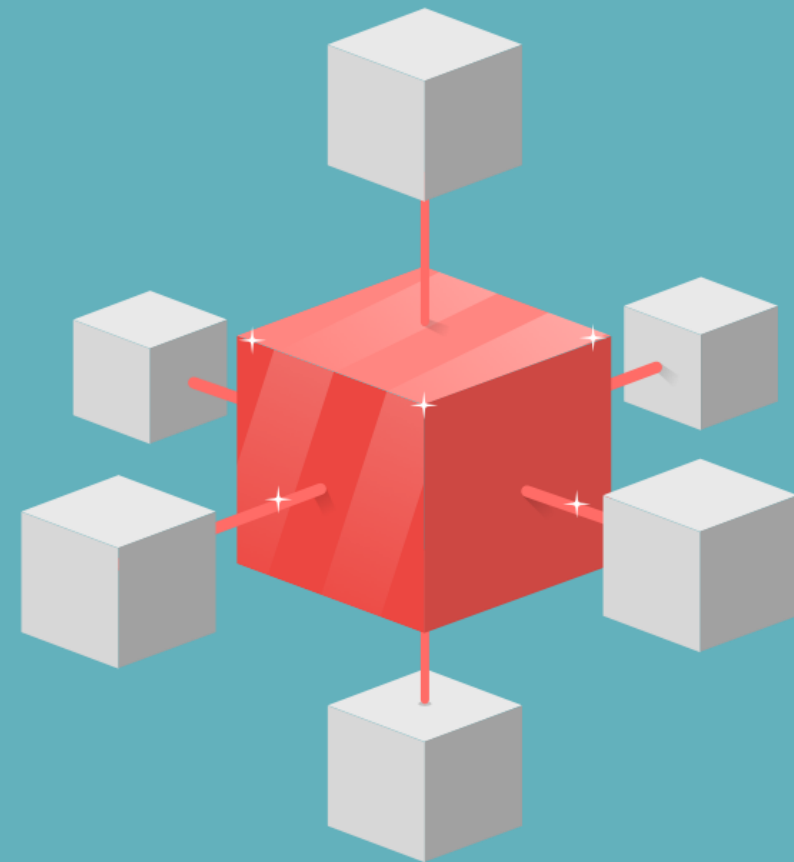
Possuem a capacidade de serem descobertos

Arquitetura de Microserviços

Em termos simples, seria a evolução natural do SOA. A arquitetura de microserviços possui um estilo onde uma aplicação é estruturada como uma coleção de pequenos serviços autônomos modelados em torno de um domínio de negócios.

Ela é utilizada para desenvolver uma aplicação com um conjunto de pequenos serviços, que funcionam com seu próprio processo.

Cada serviço é desenvolvido em torno de um conjunto de regras de negócio específicas, e é implementado de forma independente.



Arquitetura de Microserviços – vantagens

Manutenção e evolução dos serviços mais estáveis

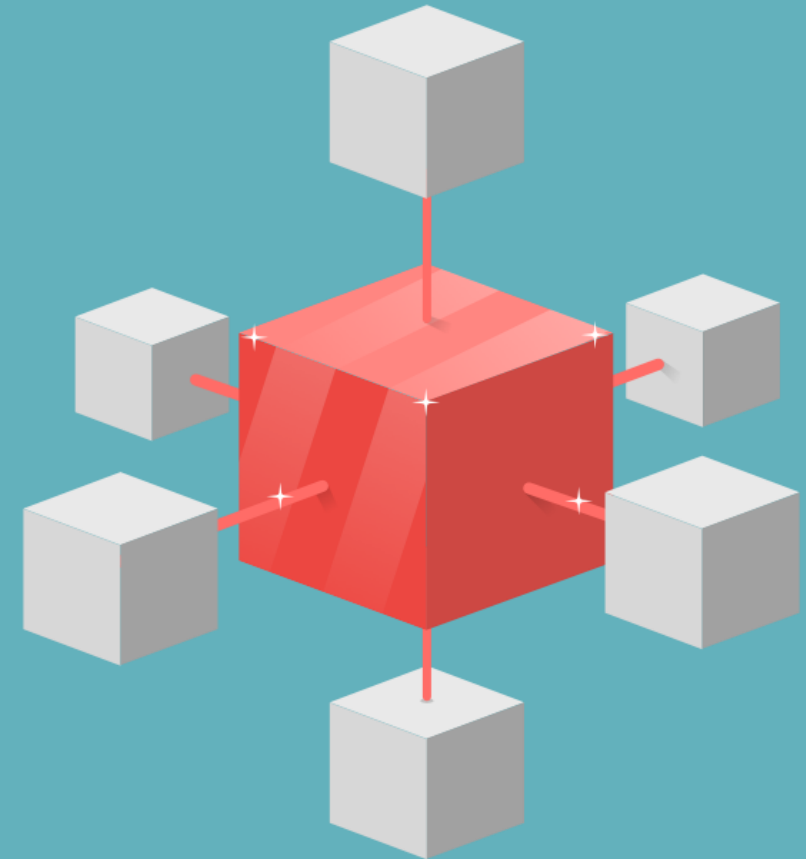
Serviços com baixo nível de acoplamento e interdependência

Escalabilidade do sistema

Redução de custos

Flexibilidade de tecnologia

Facilidade de colocar alterações em produção

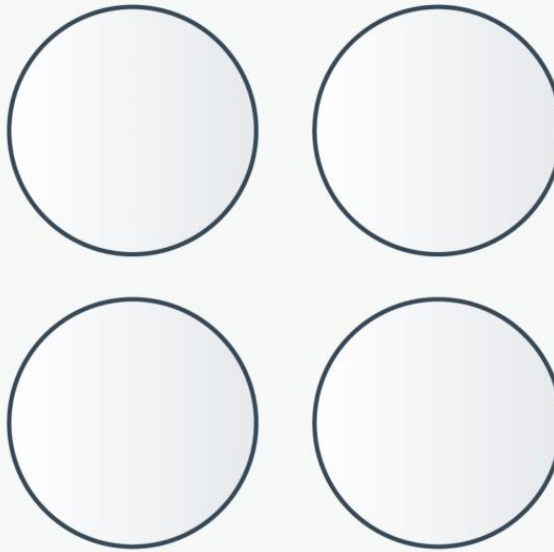


Monolithic vs. SOA vs. Microservices



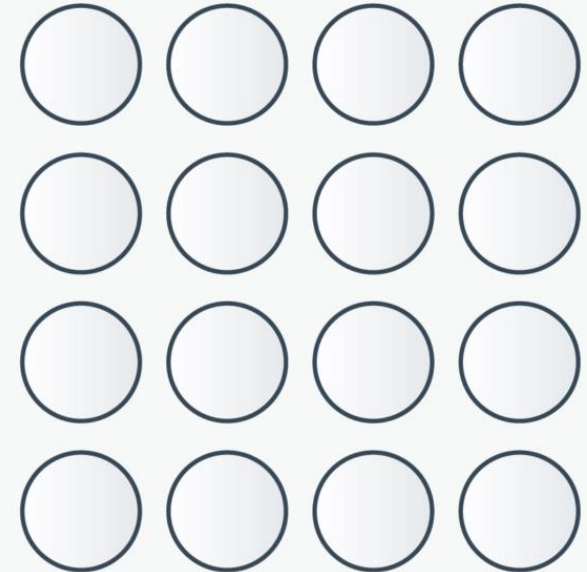
Monolithic

Single Unit



SOA

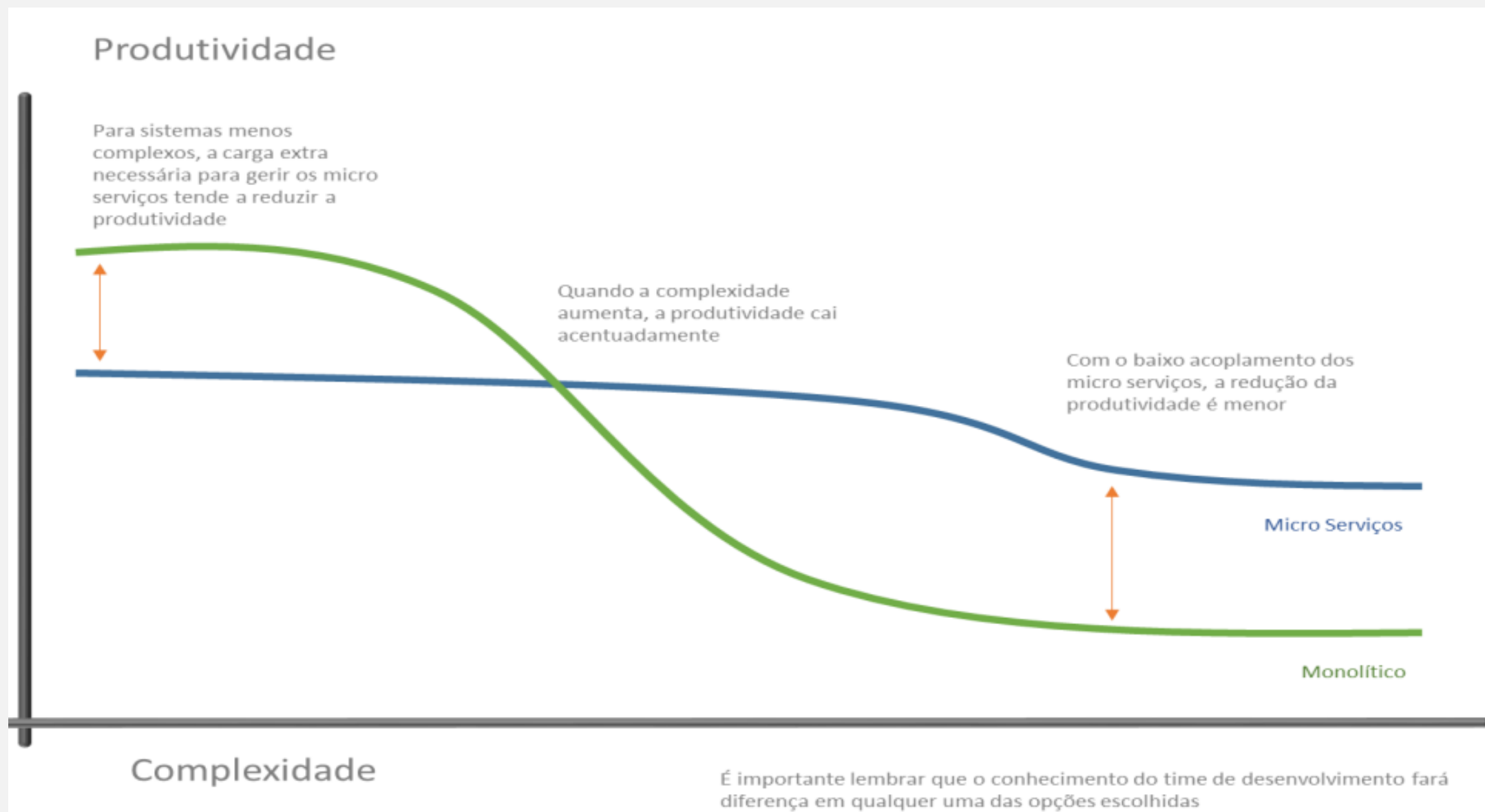
Coarse-grained



Microservices

Fine-grained

Quando utilizar um ou outro



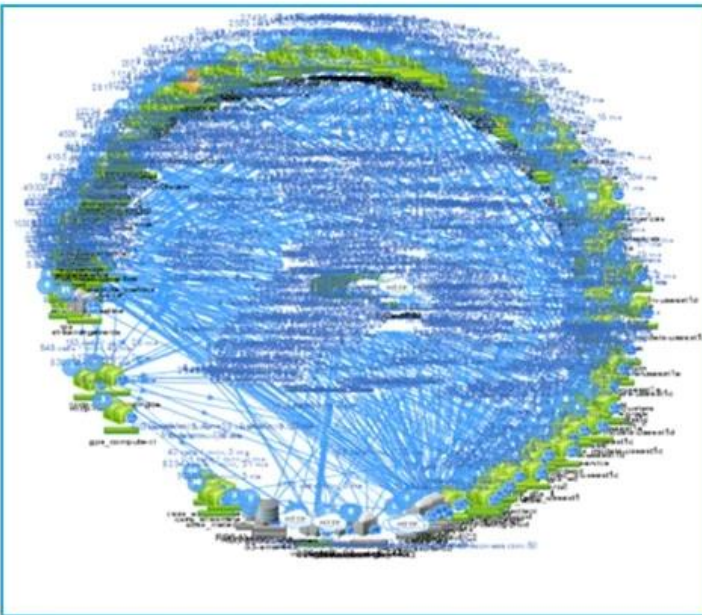
Resumo das diferenças – monolítico vs microsserviços

Categoria	Arquitetura monolítica	Arquitetura de microsserviços
Design	Base de código única com várias funções interdependentes.	Componentes de software independentes com funcionalidade autônoma que se comunicam entre si usando APIs.
Desenvolvimento	Requer menos planejamento no início, mas fica cada vez mais complexo de entender e manter.	Requer mais planejamento e infraestrutura no início, mas fica mais fácil de gerenciar e manter com o tempo.
Implantação	Aplicação inteira implantada como uma única entidade.	Cada microsserviço é uma entidade de software independente que requer implantação individual em contêineres.
Depuração	Rastreie o caminho do código no mesmo ambiente.	Requer ferramentas avançadas de depuração para rastrear a troca de dados entre vários microsserviços.

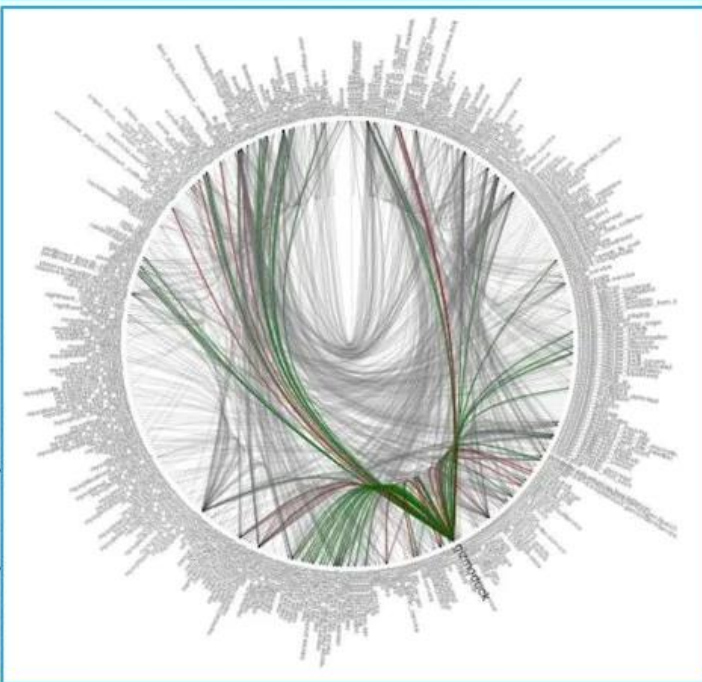
Resumo das diferenças – monolítico vs microsserviços

Categoria	Arquitetura monolítica	Arquitetura de microsserviços
Modificação	Pequenas mudanças introduzem riscos maiores, pois afetam toda a base de código.	Você pode modificar microsserviços individuais sem afetar toda a aplicação.
Escala	Você precisa escalar toda a aplicação, mesmo que apenas determinadas áreas funcionais tenham um aumento na demanda.	Você pode escalar microsserviços individuais conforme necessário, o que economiza custos gerais de escalabilidade.
Investimento	Baixo investimento inicial à custa de maiores esforços contínuos e de manutenção.	Investimento adicional de tempo e custo para configurar a infraestrutura necessária e desenvolver a competência da equipe. No entanto, economia de custos, manutenção e adaptabilidade a longo prazo.

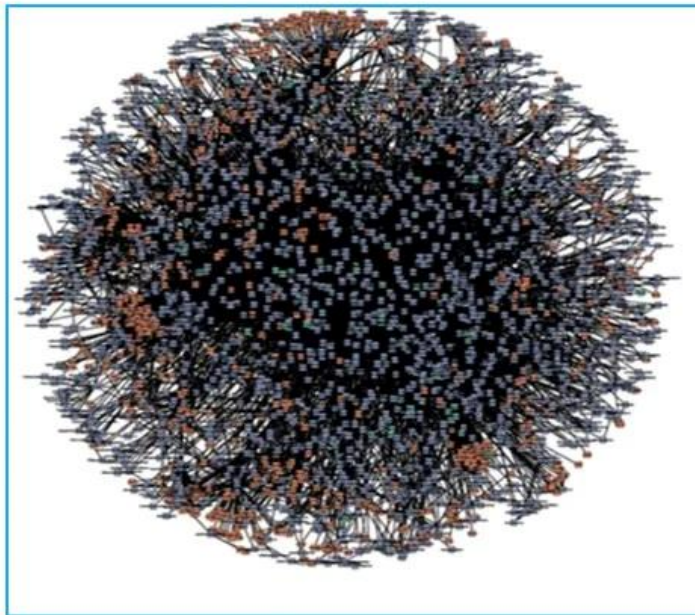
Netflix, 500+ ms



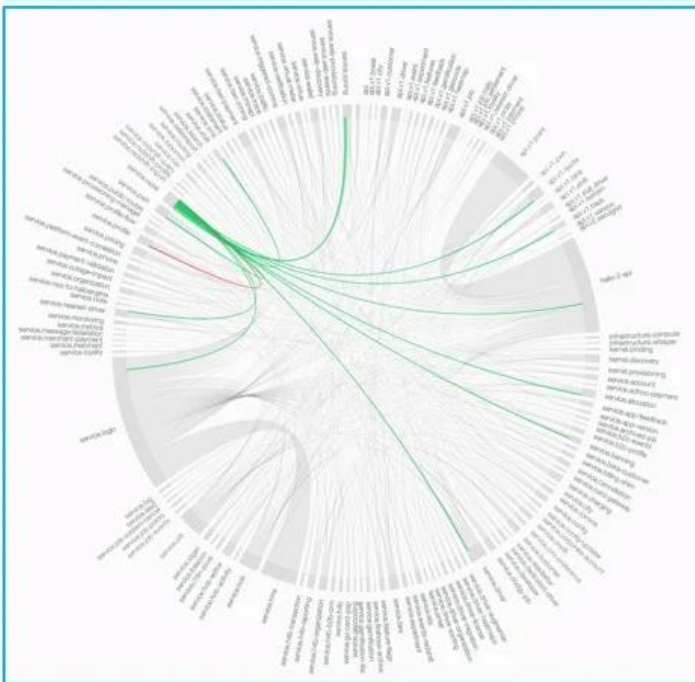
Twitter, 2013, 500+ ms



Amazon, 2009



Hailo, 450+ ms



Exemplo de grandes sistemas

As imagens exibidas são representações visuais dos sistemas de microserviços de empresas conhecidas, frequentemente referidas como **"estrela da morte"** pela sua complexidade e densa interconexão.

Elas destacam o número e as inter-relações dos microserviços utilizados pela **Netflix, Amazon, Twitter e Hailo**.

Containers

O que são containers?

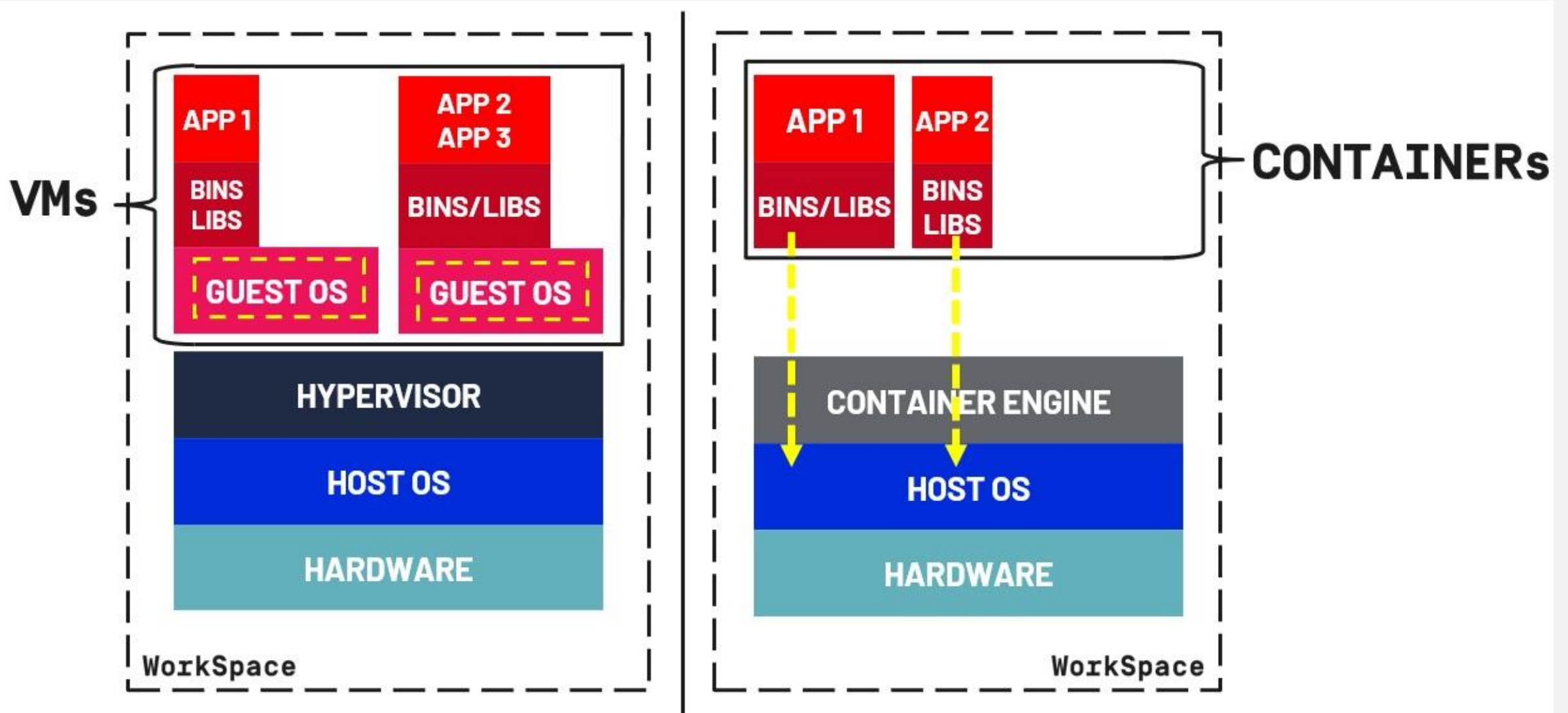
Contêineres são unidades executáveis de software em que o código do aplicativo é empacotado com suas respectivas bibliotecas e dependências, usando métodos comuns para executá-los em qualquer lugar.

Para fazer isso, os contêineres usam uma forma de virtualização de sistema operacional (SO) em que os recursos do kernel do sistema operacional podem ser utilizados para isolar processos e controlar o volume de CPU, memória e disco que podem ser acessados por esses processos.

Ao contrário de uma máquina virtual, não precisam incluir um SO guest em todas as instâncias e podem simplesmente usar os recursos do SO host.

O ano de 2013 é tido como o início da era moderna dos contêineres devido ao surgimento do **Docker**.

Virtual Machine vs Container



Benefícios dos containers

A principal vantagem dos contêineres, especialmente em comparação com uma VM, está em proporcionar um nível de abstração que os torna leves e portáteis.

- **Leve:** os contêineres compartilham o kernel do sistema operacional da máquina, eliminando a necessidade de uma instância completa do sistema operacional por aplicativo e tornando os arquivos de contêineres pequenos e fáceis de usar. O tamanho menor, especialmente em comparação com máquinas virtuais, significa que eles podem se adaptar rapidamente e suportar melhor aplicativos nativos da cloud com escala ajustada horizontalmente.
- **Móvel e independente de plataforma:** os contêineres carregam todas as dependências associadas com eles, o que significa que um software pode ser codificado uma vez e executado sem a necessidade de reconfigurá-lo nos ambientes de computação locais, na cloud ou em notebooks.

Benefícios dos containers

- **Suporte a padrões modernos de desenvolvimento e arquitetura:** devido a uma combinação de sua portabilidade/consistência de implementação entre plataformas e seu porte pequeno, os containers são ideais para desenvolvimento e padrões de aplicativos modernos, como DevOps, serverless e microsserviços, criados com implementações regulares de código em pequenos incrementos.
- **Melhor utilização:** assim como as VMs, os contêineres permitem que desenvolvedores e operadores melhorem a utilização da CPU e da memória de máquinas físicas. Um dos principais benefícios dos contêineres é que, como também permitem arquiteturas de microsserviços, os componentes de aplicativos podem ser implementados e escalados de modo mais granular, uma alternativa interessante a ter que escalar um aplicativo monolítico inteiro quando um único componente está tendo dificuldades com a carga.

Casos de uso

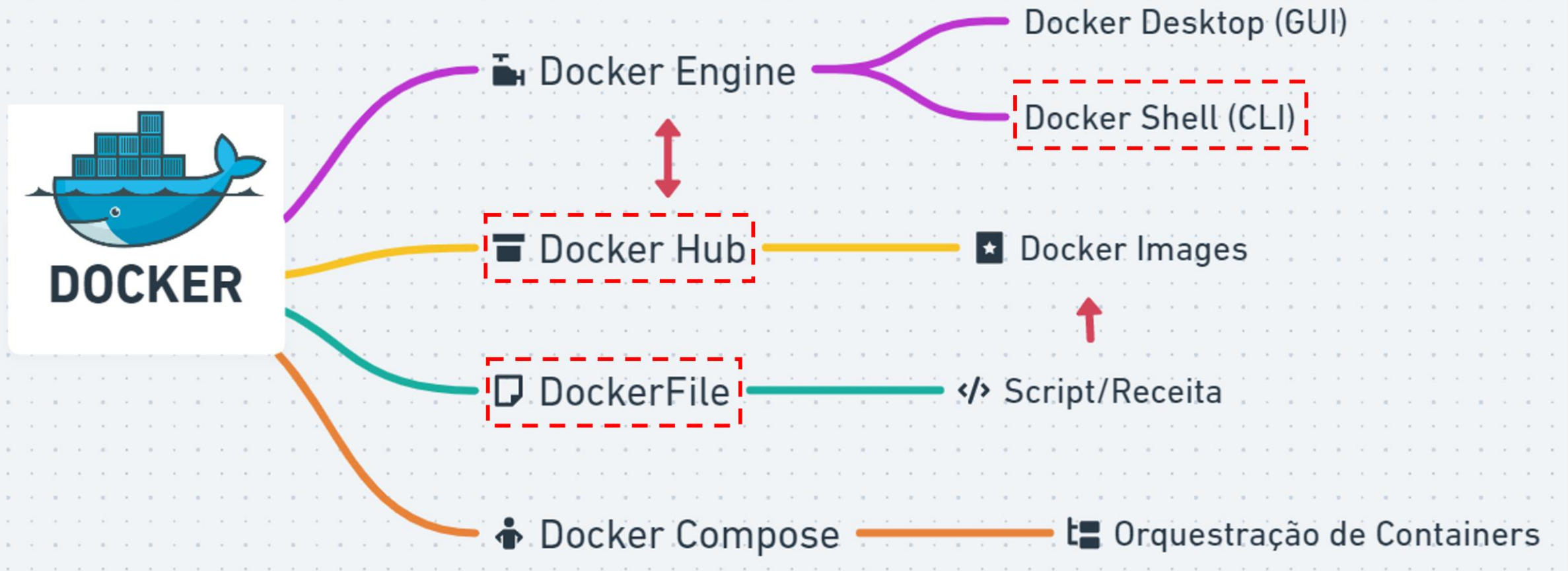
- **Microserviços:** Os containers permitem o isolamento de processos que facilitam o desmembramento e a execução de aplicativos como componentes independentes, chamados “microserviços”.
- **Processamento em lotes:** Empacote processamento em lotes e trabalhos de ETL em containers para iniciar trabalhos rapidamente e escalar os trabalhos de forma dinâmica de acordo com a demanda.
- **Machine Learning:** Use containers para escalar rapidamente modelos de Machine Learning para treinamento e inferência e executá-los perto de suas fontes de dados em qualquer plataforma.
- **Aplicativos híbridos:** Os containers permitem padronizar a maneira como o código é implantado, facilitando a criação de fluxos de trabalho para aplicativos que são executados entre ambientes locais e na nuvem.
- **Modernização e migração de aplicativos:** uma das abordagens mais comuns para a modernização de aplicativos é containerizá-los em preparação para a migração para a cloud.

Containerização

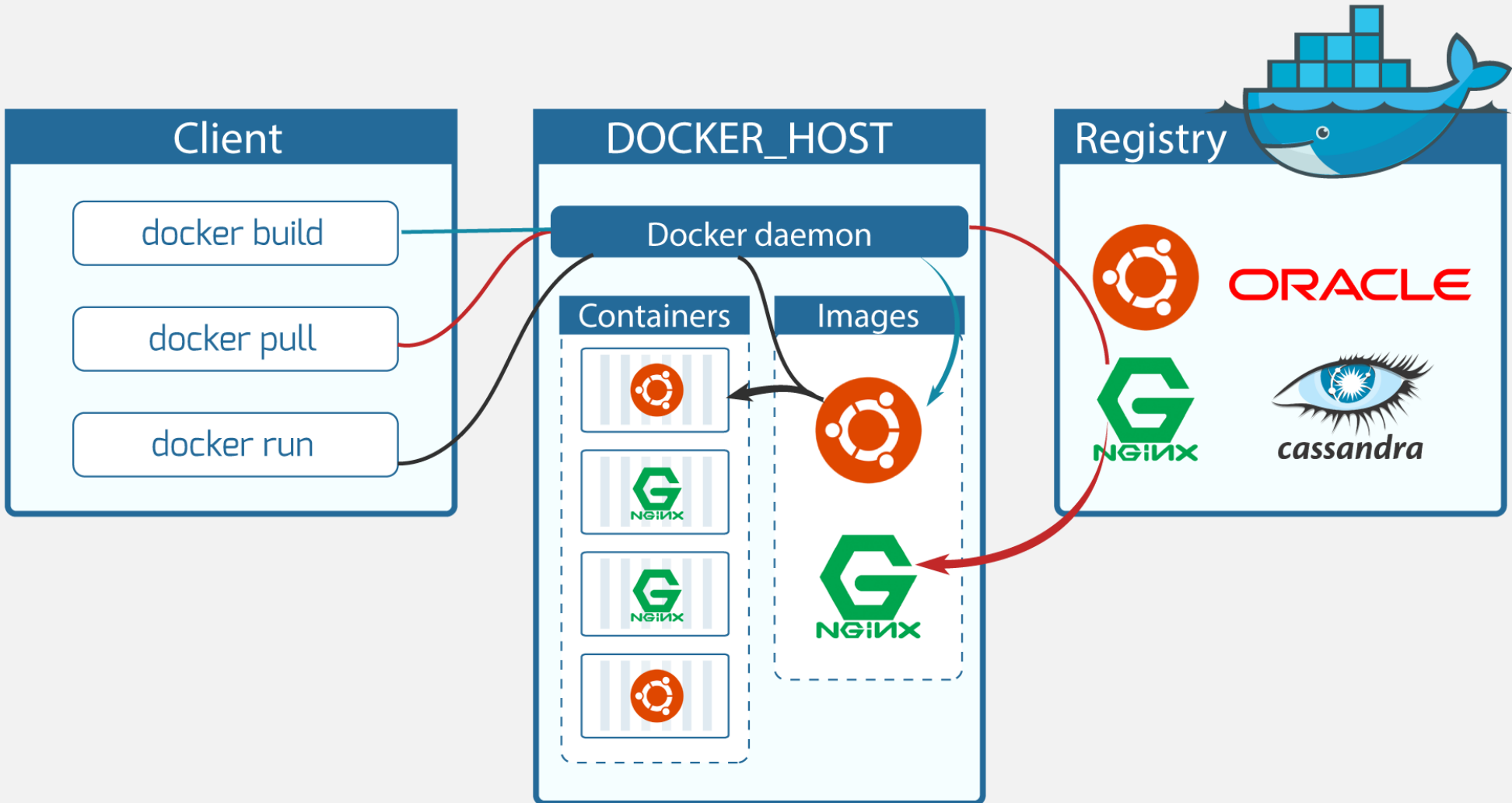
O software precisa ser projetado e empacotado de maneira diferente para aproveitar os contêineres, e esse processo é comumente chamado de containerização.

Ao containerizar um aplicativo, o processo inclui o empacotamento dele com as variáveis de ambiente, os arquivos de configuração, as bibliotecas e as dependências de software relevantes. O resultado é uma imagem de contêiner que pode ser executada em uma plataforma de contêineres.

Ferramentas Docker



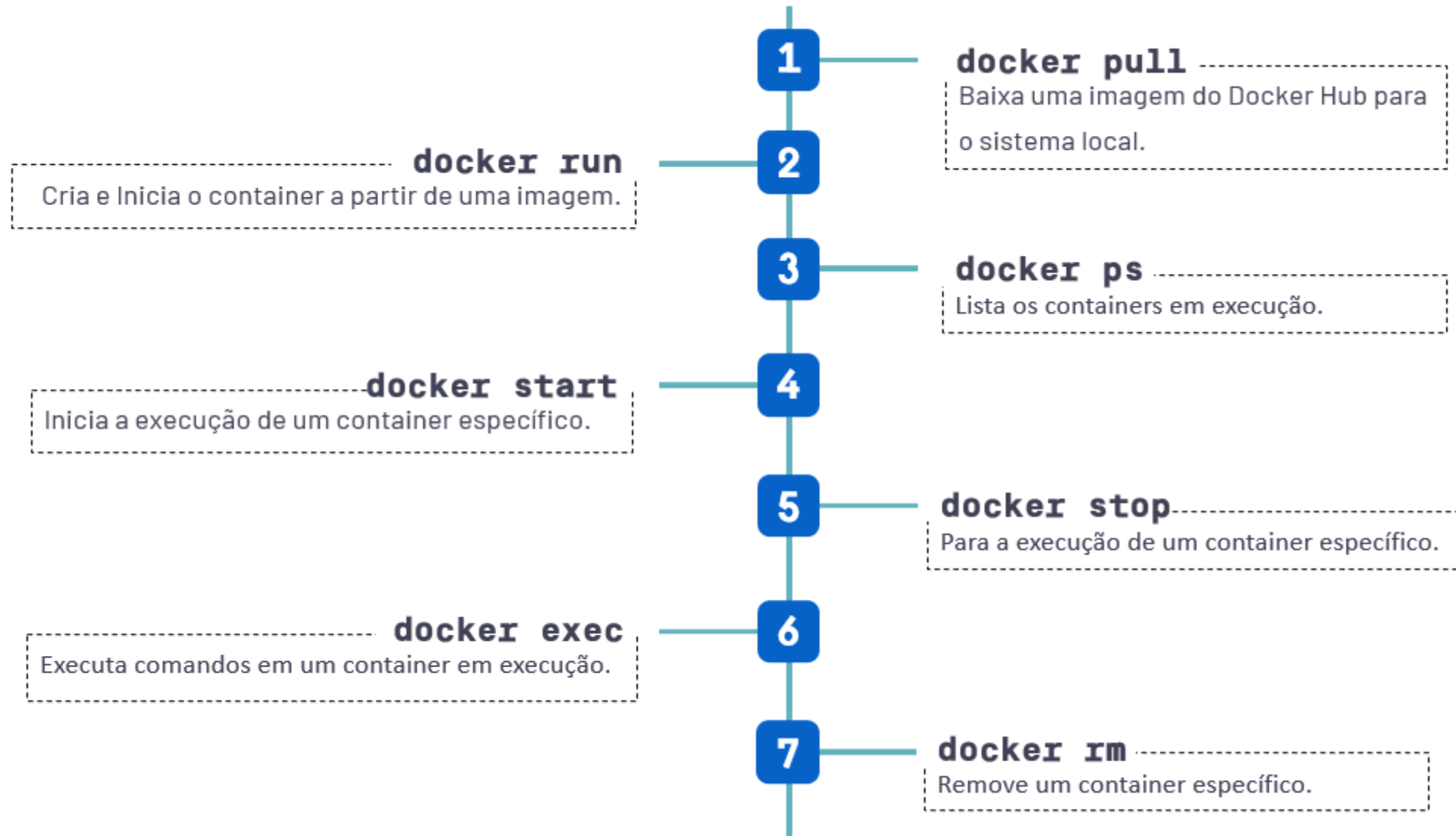
DOCKER COMPONENTS



Docker x Dockerfile x Docker Compose

- **Docker** é um mecanismo de contêiner que nos permite separar nossos aplicativos da infraestrutura em que são executados. Ele nos permite executar qualquer aplicativo em um ambiente virtualizado, usando praticamente qualquer hardware ou sistema operacional que desejarmos. Isso significa que podemos usar o mesmo ambiente para nossos aplicativos em desenvolvimento, teste e produção.
- **Dockerfile** é um arquivo de texto simples que contém instruções para construir imagens Docker. Eles seguem um padrão Dockerfile, e **o daemon Docker é o responsável final por executar o Dockerfile e gerar a imagem**. Ele pode ser baseado em um sistema operacional específico ou em uma distribuição Java. A partir daí, um Dockerfile pode realizar diversas operações para construir uma imagem.
- **Docker Compose** é uma ferramenta para definir e executar aplicativos Docker com vários contêineres. Usando um arquivo de configuração YAML, o Docker Compose nos permite configurar vários contêineres em um só lugar.


Comandos básicos Docker



Dockerfile – principais comandos

FROM - Deve ser o primeiro comando de um Dockerfile, com ele configuramos qual a imagem que queremos utilizar como base na nossa imagem.

WORKDIR - Para definirmos qual o diretório de trabalho. Assim que o container for executado, este diretório será o que iremos cair dentro ao acessar o container.

 Dockerfile X


Dockerfile

```
1 FROM python:3.8-slim-buster
2 WORKDIR ./
3 COPY requirements.txt requirements.txt
4 RUN pip install requirements.txt
5 COPY . .
6 EXPOSE 5000
7
8 CMD [ "flask", "run", "--host=0.0.0.0"]
```


COPY - Copia um arquivo ou diretório para dentro de um diretório do container.

RUN - Usado para executar comandos no contêiner, geralmente usados para instalar pacotes. Cada RUN cria uma nova camada em nosso contêiner, portanto, precisamos evitar criar RUNs demais.

EXPOSE - Configura portas que o container irá expor, e então o container poderá ser acessível por essas portas.

 Dockerfile X


Dockerfile exemplo

 Dockerfile


```
1 FROM python:3.8-slim-buster
2 WORKDIR ./
3 COPY requirements.txt requirements.txt
4 RUN pip install requirements.txt
5 COPY . .
6 EXPOSE 5000
7
8 CMD [ "flask", "run", "--host=0.0.0.0"]
```

CMD - Define um comando para ser executado quando um container com esta imagem inicializar.

Pode haver apenas uma instrução CMD em um Dockerfile. Se você adicionar mais de um, apenas o último entrará em vigor..

 Dockerfile X

Dockerfile exemplo

 Dockerfile

```
1 FROM python:3.8-slim-buster
2 WORKDIR ./
3 COPY requirements.txt requirements.txt
4 RUN pip install requirements.txt
5 COPY . .
6 EXPOSE 5000
7
8 CMD [ "flask", "run", "--host=0.0.0.0"]
```

Build container

```
docker build -t <nome_da_imagem> .
```

Agora para buildar a imagem, usamos o comando build do Docker. passamos o parametro -t para nomear esta imagem, dois pontos e uma versão. E utilizamos o ".", para dizer que nosso Dockerfile se encontra neste mesmo nível de diretório. Não apontamos o Dockerfile, e sim o diretório que ele se encontra. E com o comando DOCKER RUN criamos e iniciamos o container!



Dockerfile U x

Dockerfile > ...

```
1 FROM maven:3.6-openjdk-17 AS build
2
3 WORKDIR /builder
4
5 COPY . /builder/
6
7 RUN mvn clean package
8
9 FROM amazoncorretto:17-alpine3.16
10
11 WORKDIR /app
12
13 COPY --from=build /builder/target/*.jar /app/app.jar
14
15 CMD ["java", "-jar", "app.jar"]
16
```

Dockerfile java Multi-stage Build

Subindo a imagem Docker no Docker Hub

Imagem ou Container?

Traçando um paralelo com o conceito de orientação a objeto, a imagem é a classe e o container o objeto. A imagem é a abstração da infraestrutura em estado somente leitura, de onde será instanciado o container.

Todo container é iniciado a partir de uma imagem, dessa forma podemos concluir que nunca teremos uma imagem em execução.

Um container só pode ser iniciado a partir de uma única imagem. Caso deseje um comportamento diferente, será necessário customizar a imagem.

O que é Docker hub?

Dockerhub é um Serviço de Web Hosting compartilhado para imagens Docker. É no Dockerhub que os Dockerfiles são disponibilizadas para a comunidade. imagens oficiais são disponibilizadas por grandes grupos para facilitar o seu trabalho na hora de criar uma infraestrutura Docker.

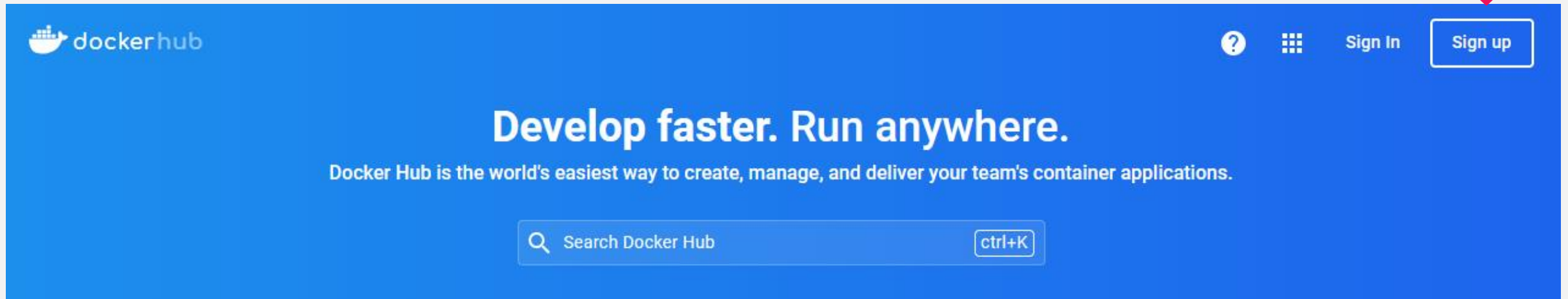
Ao necessitar de um container com um sistema de gerenciamento de Banco de Dados por exemplo, você pode usar a imagem oficial do MySQL que é mantido pela própria Docker.

Diversas tecnologias também possuem imagens oficiais disponíveis na plataforma:

- Python
- Node
- Nginx

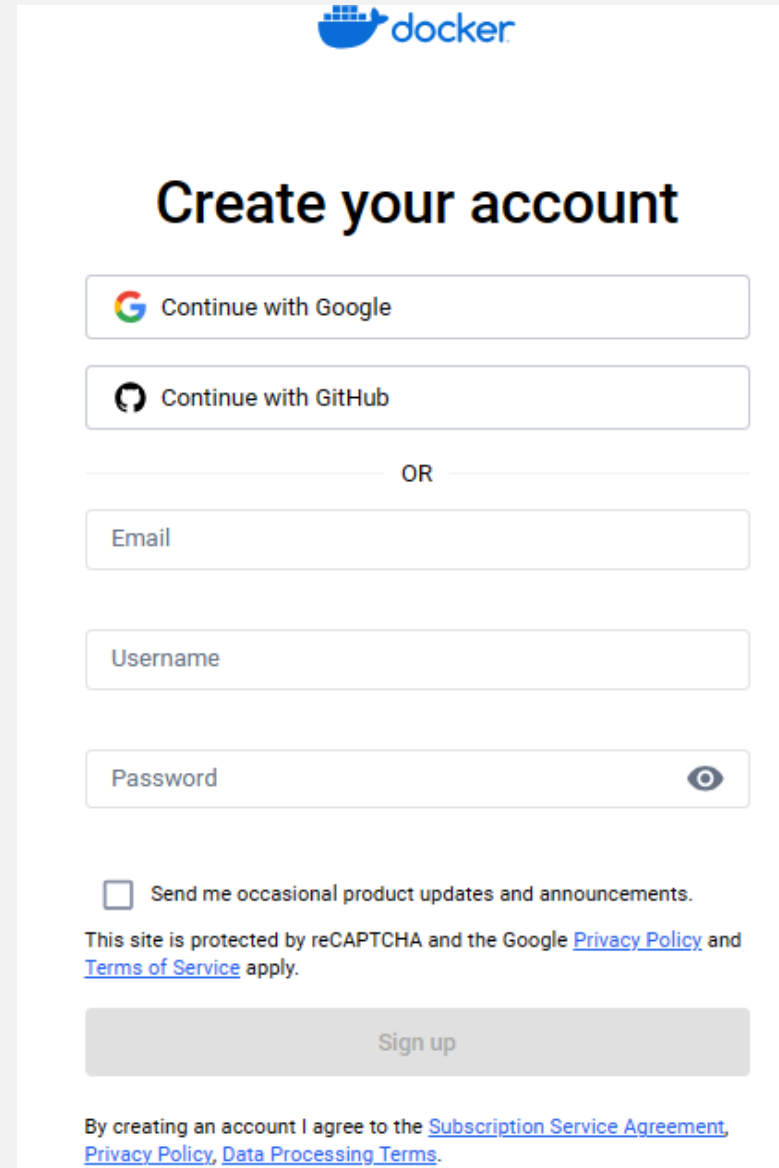
Criando sua conta e um repositório

Acesse [Docker Hub Container Image Library | App Containerization](#)




Criando sua conta e um repositório


Você pode se logar com algum e-mail do google, sua conta GitHub ou até se cadastrar com qualquer outro e-mail




The screenshot shows the Docker account creation interface. At the top is the Docker logo. Below it is the heading "Create your account". There are two buttons for social login: "Continue with Google" and "Continue with GitHub". Below these is a horizontal line with "OR" in the center. Then there are three input fields: "Email", "Username", and "Password". The "Password" field has an eye icon to toggle visibility. Below the input fields is a checkbox labeled "Send me occasional product updates and announcements." At the bottom of the form area is a "Sign up" button. Below the button is a line of text stating: "This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply." At the very bottom, there is a line of text: "By creating an account I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), [Data Processing Terms](#)."

 docker

Create your account


 Continue with Google

 Continue with GitHub

OR

Email

Username

Password 

☐ Send me occasional product updates and announcements.

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

Sign up

By creating an account I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), [Data Processing Terms](#).

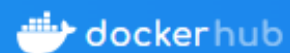
Criando sua conta e um repositório

Após se logar, o Docker Hub te encaminhará para o seu perfil. Nessa página clique em **Create Repository** e preencha um pequeno formulário com o nome do repositório e uma breve descrição, certifique-se de que o repositório esteja como público para que outras pessoas possam visualizar a sua imagem.

A conta gratuita possui direito a um repositório privado e ilimitados repositórios públicos. Há planos para que mais repositórios privados fiquem disponíveis, geralmente voltado para empresas que usam Docker em sua infraestrutura.

Agora que você já tem a sua conta e criou um repositório vamos enviar a sua imagem para o repositório.

Criando sua conta e um repositório

[Explore](#)[Repositories](#)[Organizations](#)[ctrl+K](#)[E](#)[Repositories](#) / [Create](#)Using 0 of 1 private repositories. [Get more](#)

Create repository

Namespace

eduverri

Repository Name *



Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public

Appears in Docker Hub search results



Private

Only visible to you

[Cancel](#)[Create](#)

Pushing images

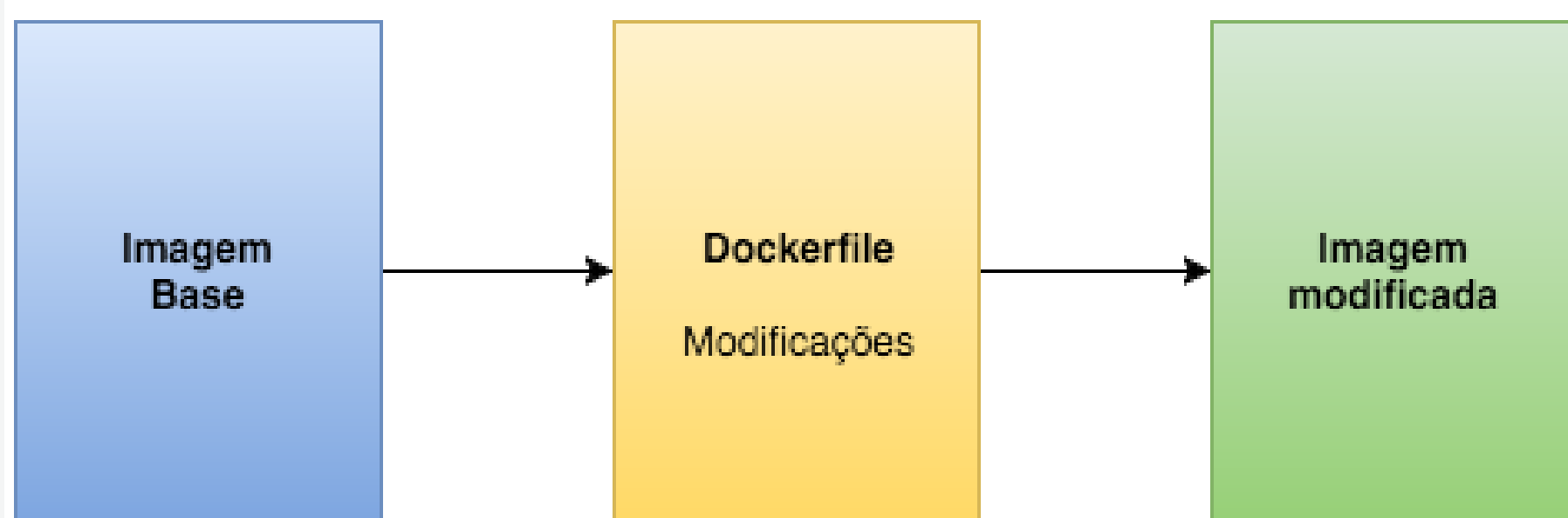
You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Criando imagens com Dockerfile

- Quando se utiliza Dockerfile para gerar uma imagem, basicamente, é apresentada uma lista de instruções que serão aplicadas em determinada imagem para que outra imagem seja gerada com base nas modificações.
- Ao utilizar o comando **docker build -t <tag_imagem>**, todos os arquivos da pasta atual serão enviados para o serviço do docker e apenas eles podem ser usados para manipulações do Dockerfile



Taggeando a sua imagem

- Antes de enviarmos nossa imagem para a nuvem precisamos taggea-la com o comando docker tag.
- docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]**
- A tag deve ser ASCII válida e pode conter letras maiúsculas e minúsculas, dígitos, sublinhados, pontos e hifens. Não pode começar com ponto final ou hífen e não deve ter mais de 128 caracteres.
- Se você não especificar uma tag, o comando usará **latest** por padrão.

Tells your operating system you are using the **docker** program

docker

tag

A *subcommand* that tags an image.

The image ID

7d9495d03763

The image name.

maryatdocker/docker-whale:latest

Your account name from Docker Hub.

Version label or tag.

Como dar push na imagem

- Antes de aprendermos como empurrar nossa imagem para o Dockerhub precisamos fazer login com a conta que nós criamos. Abra o terminal e digite o comando **docker login**, ele não aceita parâmetros mas solicita o seu usuário e senha para realizar o login como no exemplo abaixo
- Após o login realizado com sucesso podemos enviar nossa imagem utilizando o comando **docker push**

```
docker login
```

```
Username: *****  
Password: *****  
Login Succeeded
```

```
docker push dockerID/docker-is-cool
```

```
The push refers to a repository [dockerID/docker-is-cool] (len: 1)  
8d9495d05463: Image already exists  
...  
e9e06b06e14c: Image successfully pushed  
Digest:  
sha256:ad89e88beb7dc73bf55d456e2c600e0a39dd6c9500d7cd8d1025626c4b9850  
11
```


Comandos para usar com sua imagem

- **docker pull seu_id/seu_repo:tag** baixa a sua imagem do Docker hub para a máquina
- **docker image rm seu_id/seu_repo:tag** remove a imagem baixada
- **docker run -d -p port_host:port_docker --name nome_container imagem_container**
 - -d: roda o container de forma *detached* permitindo liberar o terminal após inicializar o container
 - -p: determina o expose de portas entre a máquina host e o container
- **docker run -d -p 5000:5000 --name teste eduverri/devops:2.0** por exemplo, roda um container utilizando uma imagem que está em meu repositório de nome eduverri/devops com a tag de 2.0 na porta 5000 da máquina host e 5000 no container, de forma detached e com um nome "teste"

Atividade valendo nota

Precisaremos do Docker daqui pra frente

Sistemas Operacionais Suportados: Docker é otimizado para sistemas com kernel Linux. Recomenda-se utilizar uma distribuição Linux para uma experiência completa.

Usuários de Windows: Para utilizar o Docker em Windows, a melhor opção é através do WSL2, instalando uma distribuição Linux, como Ubuntu, dentro do WSL2.

Método de Instalação Recomendado:

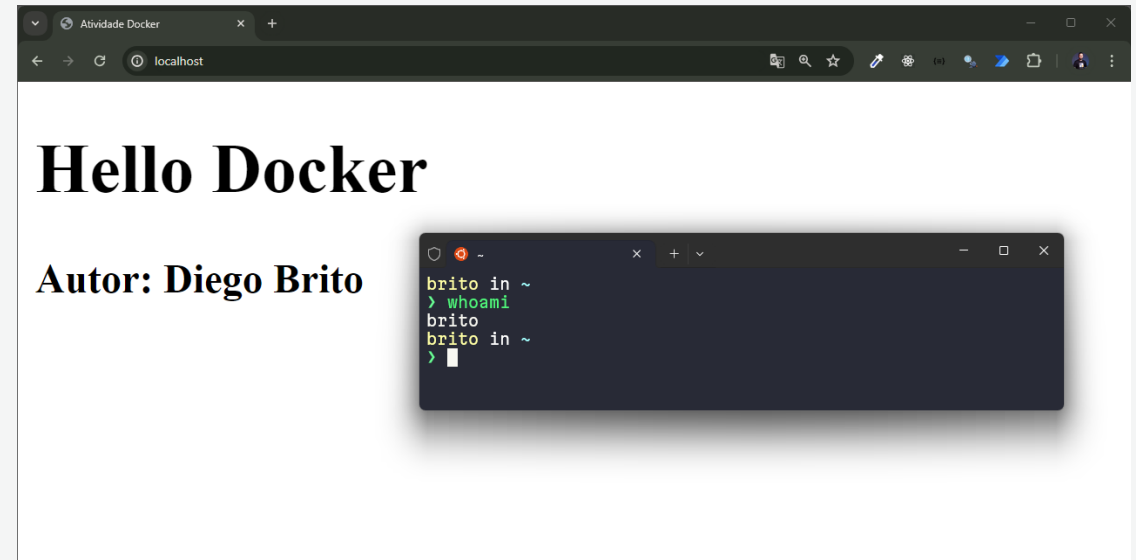
- Prefira a instalação do Docker Engine diretamente via linha de comando.
- Evite o uso do Docker Desktop, pois oferece menor desempenho e depende de interface gráfica, que pode não estar sempre disponível.

[Instalação Docker Engine aqui.](#)

Atividade Docker hub – Passo 1

Localmente

- Criar um Dockerfile para construir uma imagem simples com um servidor web (**NGINX + index.html**);
- Construir uma imagem Docker localmente;
- Print da imagem rodando localmente + terminal aberto com o comando whoami (bash);
- Fazer login no Docker hub e criar um repositório;
- Taggear a imagem com nome e versão;
- Realizar um Docker push no Docker Hub;
- Anexar link da imagem na entrega;



Atividade Docker hub – Passo 2

Na VM de sub-rede pública (AWS)

- Realizar um Docker pull dessa imagem;
- Checar as imagens Docker na máquina;
- Realizar um Docker run com a imagem baixada;
- Testar o servidor web;
- Print do navegador com container executando na VM + terminal aberto com o comando whoami (bash) – vide passo 1;

Atividade Docker hub – dica básico

Estrutura local

```
urubu100@sptech02608:~$ tree ./project_docker
./project_docker
├── Dockerfile
└── index.html

0 directories, 2 files
```

Dockerfile

```
FROM nginx
COPY index.html /usr/share/nginx/html
```

HTML

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Meu Primeiro Container Docker!
</title>
</head>
<body>
  <h1>Olá, mundo!</h1>
  <p>Este é meu primeiro container</p>
</body>
</html>
```

```
docker build -t eduverri/devops:nginx .
```

```
docker run -p 80:80 --name docker-nginx eduverri/devops:nginx
```

Agradeço
a sua atenção!



SÃO
PAULO
TECH
SCHOOL