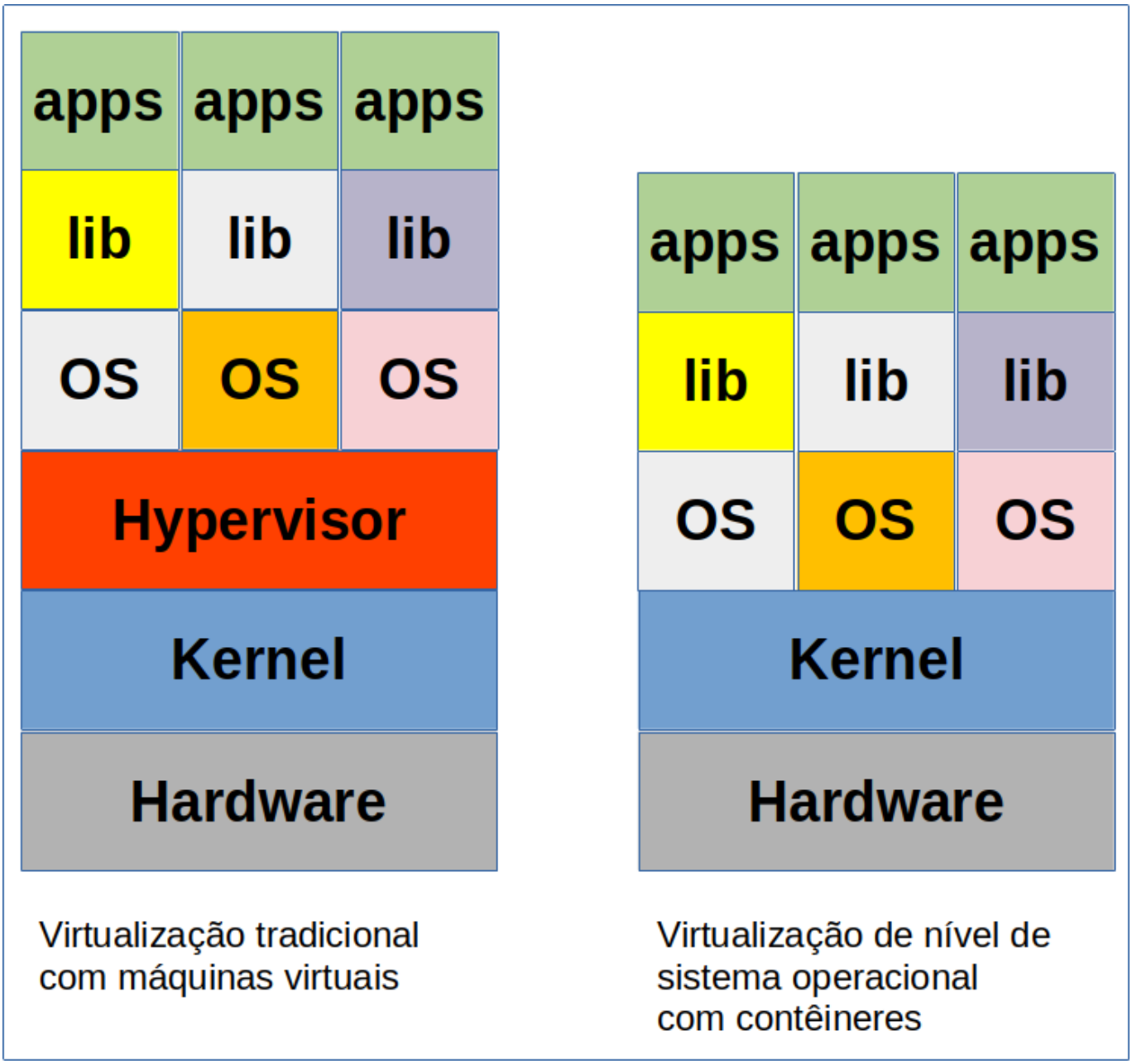
Deploy de aplicações com Docker, Kubernetes e Terraform

Introdução ao Docker

LXC - Linux Containers

Linux Containers (**LXC**) é uma tecnologia de virtualização leve que permite a criação e execução de ambientes isolados no sistema operacional Linux. Esses ambientes isolados, conhecidos como contêineres, oferecem uma maneira eficiente de empacotar e distribuir aplicativos, juntamente com suas dependências, em uma unidade autônoma. Os contêineres LXC compartilham o kernel do sistema host, o que os torna mais leves e rápidos do que as máquinas virtuais tradicionais.



A relação entre LXC e **Docker** é estreita. Docker inicialmente utilizou a tecnologia LXC para criar e gerenciar contêineres. No entanto, com o tempo, o Docker desenvolveu sua própria biblioteca, chamada libcontainer, para interagir diretamente com o kernel do Linux, eliminando a dependência do LXC. Isso permitiu ao Docker se tornar uma plataforma de contêineres mais flexível e portátil, independentemente do sistema de gerenciamento de contêineres subjacente.

Aqui estão algumas diferenças chave entre LXC e Docker:

1. **Nível de abstração**: LXC fornece uma abstração mais baixa, permitindo maior controle sobre o sistema de contêineres. Docker oferece uma camada de abstração mais alta, simplificando o processo de criação e execução de contêineres.

2. **Integração de aplicativos**: Docker é mais orientado para a construção e distribuição de aplicativos encapsulados, enquanto LXC é mais voltado para a criação de ambientes isolados para executar aplicativos.

3. **Ecossistema**: Docker possui um ecossistema robusto de ferramentas e serviços relacionados, como **Docker Compose** e **Docker Swarm**, que facilitam a orquestração de contêineres em larga escala. LXC

tem menos ferramentas específicas associadas a ele.

Em resumo, LXC é uma tecnologia de virtualização de nível mais baixo que pode ser usada para criar ambientes isolados no Linux. Docker, por outro lado, é uma plataforma de contêineres que construiu sua própria camada de abstração sobre o LXC, tornando mais fácil para os desenvolvedores criar, distribuir e executar aplicativos em contêineres. Ambos têm seus usos e vantagens, dependendo das necessidades específicas de um projeto.

> Posso usar LXC no deploy das minhas aplicações?

A princípio, pode sim. É claro que contará com menos ferramentas e utilitários, podendo ter alguma dificuldade para configurar seus conteineres, mas pode sim.

Docker

Docker é uma plataforma de software que revolucionou a maneira como os aplicativos são construídos, enviados e executados, usando a tecnologia de contêineres. Vamos explorar um pouco da história do Docker e seus componentes principais:

História do Docker

1. **Origem (2013):** Docker foi introduzido pela primeira vez em 2013 pela empresa Docker, Inc. (originalmente dotCloud), fundada por Solomon Hykes. O Docker expandiu e popularizou o conceito de contêinerização no desenvolvimento de software.

2. **Contêineres vs. Máquinas Virtuais:** Antes do Docker, a virtualização era amplamente realizada através de máquinas virtuais (VMs), que emulam hardware e executam um sistema operacional completo. Docker introduziu uma abordagem mais leve usando contêineres, que compartilham o mesmo kernel do sistema operacional do host, mas isolam os processos do aplicativo, tornando-os mais eficientes em termos de recursos.

3. **Crescimento e Popularidade:** A adoção do Docker cresceu rapidamente devido à sua facilidade de uso, portabilidade e eficiência. Ele se tornou um padrão de facto para a contêinerização, impulsionando o desenvolvimento de microserviços e a arquitetura orientada a contêineres.

4. **Open Source:** Docker é um projeto de código aberto, o que contribuiu significativamente para sua adoção e evolução. A comunidade de código aberto desempenhou um papel crucial no desenvolvimento de novos recursos e na manutenção do ecossistema Docker.

Componentes Principais do Docker

1. **Docker Engine:** É o componente central que cria e executa os contêineres Docker. Ele consiste em um daemon de servidor, uma API REST para interação e uma interface de linha de comando (CLI) para interagir com o daemon.

2. **Imagens Docker:** São os modelos de leitura apenas usados para criar contêineres. Eles incluem o código do aplicativo, bibliotecas, dependências e outras instruções necessárias para executar o aplicativo.

3. **Contêineres Docker:** São as instâncias em tempo de execução de imagens Docker. Eles encapsulam o aplicativo e seu ambiente. Várias instâncias de contêineres podem ser criadas a partir da mesma imagem.

4. **Dockerfile:** É um script de configuração que contém uma série de instruções para construir uma imagem Docker. Ele simplifica o processo de automação e garante a consistência entre os ambientes de desenvolvimento e produção.

5. **Docker Compose:** Uma ferramenta para definir e executar aplicativos Docker multi-contêiner. Permite que você use um arquivo YAML para configurar os serviços do aplicativo e simplifica o processo de implantação em ambientes complexos.

6. **Docker Hub e Registro:** Docker Hub é um serviço de registro que permite armazenar e compartilhar imagens Docker. Os usuários podem baixar imagens públicas ou armazenar e compartilhar suas próprias.

7. **Orquestração:** Ferramentas como Docker Swarm e Kubernetes podem ser usadas para gerenciar, escalar e orquestrar contêineres Docker em clusters de servidores.

Impacto e Evolução

- **Influência na Indústria:** Docker influenciou significativamente o desenvolvimento de aplicativos, CI/CD (Integração Contínua e Entrega Contínua), e impulsionou a adoção da arquitetura de microserviços.
- **Evolução Contínua:** Docker continua evoluindo, com foco em segurança, escalabilidade e integração com outras ferramentas e plataformas.

O Docker, com sua abordagem inovadora para contêineres, mudou a paisagem do desenvolvimento de software, tornando-o mais eficiente, portátil e escalável.

Instalação do Docker

Instalar o Docker em diferentes sistemas operacionais envolve processos específicos para cada um. Vamos abordar como instalar o Docker no Linux, Windows e macOS:

No Linux

A instalação no Linux varia de acordo com a distribuição. Aqui estão as etapas gerais para Ubuntu, que é uma das distribuições mais populares:

1. **Atualize o Índice de Pacotes:**

```
sudo apt-get update
```

2. **Instale Pacotes Necessários:**

```
sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common
```

3. **Adicione a Chave GPG Oficial do Docker:**

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
```

4. **Adicione o Repositório do Docker às Fontes do APT:**

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

5. **Atualize o Índice de Pacotes Novamente:**

```
sudo apt-get update
```

6. **Instale o Docker:**

```
sudo apt-get install docker-ce
```

7. **Verifique a Instalação:**

```
sudo systemctl status docker
```

No Windows

Para o Windows, o Docker oferece o Docker Desktop, que inclui Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes e Credential Helper.

1. **Baixe o Docker Desktop:** Vá para o site oficial do Docker e baixe o Docker Desktop para Windows.
2. **Execute o Instalador:** Siga as instruções no assistente de instalação.
3. **Reinicie o Sistema:** Após a instalação, reinicie o seu computador.
4. **Execute o Docker Desktop:** Após reiniciar, abra o Docker Desktop. Ele pode pedir permissões adicionais ou configurações.
5. **Verifique a Instalação:** Abra um terminal e execute `docker --version` para verificar a instalação.

No MacOS

Assim como no Windows, o Docker Desktop está disponível para macOS.

1. **Baixe o Docker Desktop:** Vá para o site oficial do Docker e baixe o Docker Desktop para macOS.
2. **Abra o Arquivo DMG:** Após o download, abra o arquivo `.dmg` e arraste o Docker para a pasta de Aplicações.
3. **Execute o Docker Desktop:** Abra o Docker a partir da pasta de Aplicações.
4. **Conceda Permissões:** Se solicitado, forneça as permissões necessárias.
5. **Verifique a Instalação:** Abra um terminal e execute `docker --version` para confirmar a instalação.

Notas Adicionais

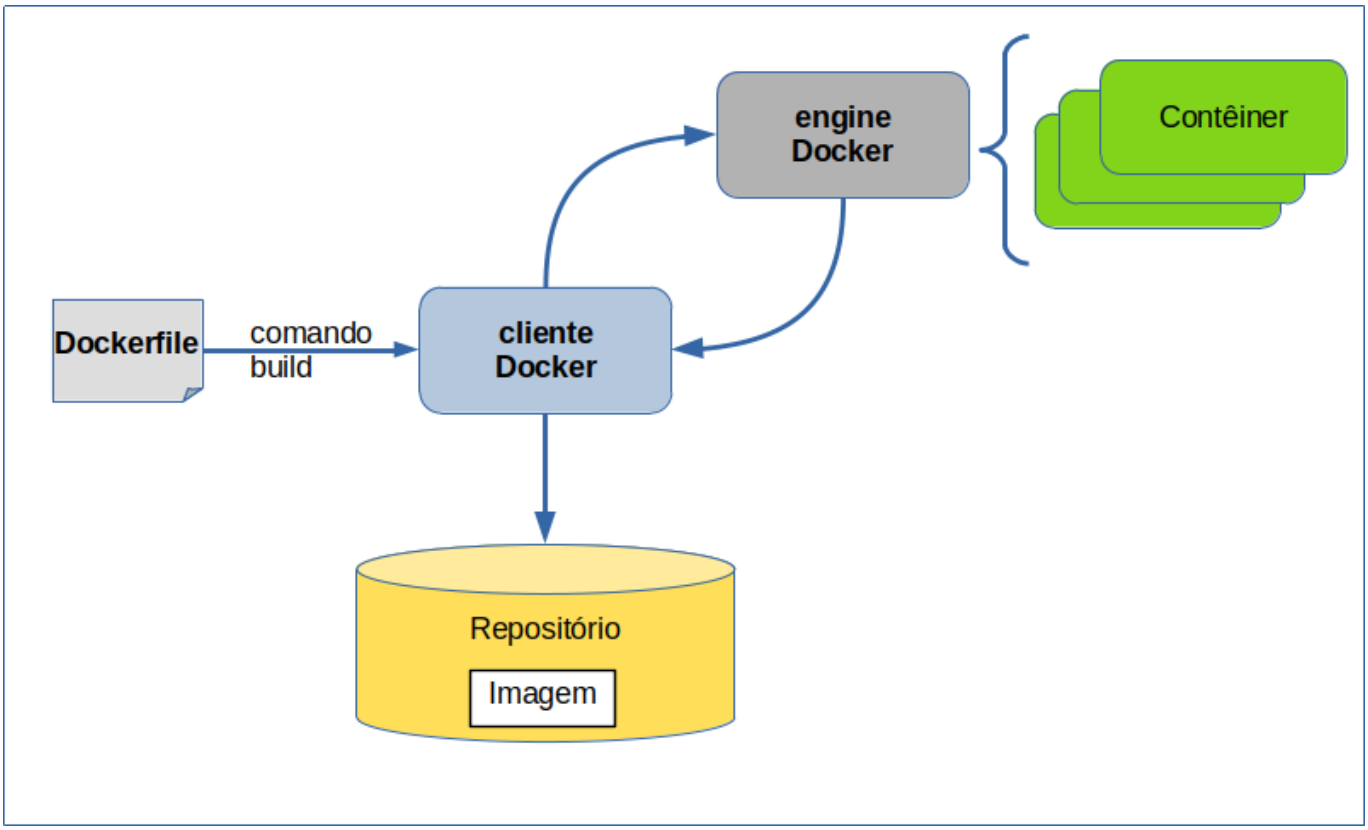
- No Linux, as etapas variam um pouco entre distribuições diferentes (como Fedora, CentOS, etc.), então sempre verifique a documentação específica para sua distribuição.
- Certifique-se de que seu sistema atenda aos requisitos mínimos de hardware e software para o Docker Desktop no Windows e macOS.
- Para Windows, é necessário ter o WSL 2 (Windows Subsystem for Linux) instalado para versões mais recentes do Docker Desktop.
- Se estiver instalando em Linux, o uso do comando **docker** exigirá permissão de **root** (sudo). Se não quiser ficar digitando "sudo" a todo momento, então crie um grupo **docker** e adicione seu usuário a ele. Em algumas distros, a instalação já faz isso. Exemplo:

```

```

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

Imagem



Uma imagem Docker, criada a partir de um Dockerfile, é um dos conceitos centrais na tecnologia Docker. Vamos explorar o que isso significa e onde essas imagens residem após serem construídas:

O que é uma Imagem Docker?

1. **Definição:** Uma imagem Docker é um modelo imutável e de somente leitura usado para criar um contêiner Docker. Ela contém o código do aplicativo, bibliotecas, dependências e outras instruções necessárias para executar o aplicativo em um ambiente Docker.

2. **Camadas de Imagem:** Uma imagem Docker é composta de várias camadas. Cada camada representa uma instrução no Dockerfile. Quando uma imagem é alterada, apenas as camadas alteradas são reconstruídas, o que torna o processo de construção rápido e eficiente.

Dockerfile

1. **Definição:** Um Dockerfile é um script de texto que contém instruções para construir uma imagem Docker. Ele define o ambiente necessário, incluindo a base a partir da qual a imagem é construída, os comandos a serem executados, os arquivos a serem adicionados e outras configurações.

2. **Processo de Construção:** Quando você executa um comando de construção (como `docker build`), o Docker lê o Dockerfile, executa as instruções nele e cria uma imagem Docker como resultado.

Onde as Imagens Residem Após o Build?

1. **Localmente:** Após uma imagem Docker ser construída a partir de um Dockerfile, ela é armazenada localmente no sistema do usuário. Você pode visualizar essas imagens locais usando o comando `docker images`.

2. **Registros Docker:** Frequentemente, as imagens são enviadas para um registro Docker, como o Docker Hub, para armazenamento, compartilhamento e versionamento. Isso é feito através do comando `docker push`. Uma vez no registro, as imagens podem ser baixadas (`docker pull`) e usadas por qualquer pessoa com acesso, dependendo das configurações de privacidade.

3. **Camadas de Imagens Compartilhadas:** Se diferentes imagens usam as mesmas camadas (por exemplo, a mesma imagem base), essas camadas não são duplicadas. Isso economiza espaço e torna o gerenciamento de imagens mais eficiente.

4. **Gerenciamento de Imagens:** As imagens Docker podem ser gerenciadas localmente ou em um registro, incluindo a remoção de imagens não utilizadas para liberar espaço.

Pontos importantes

- A criação de uma imagem Docker a partir de um Dockerfile é um processo fundamental para a implementação eficiente de aplicativos em ambientes Docker.
- Essas imagens são armazenadas localmente após a construção e podem ser enviadas para registros remotos para fácil acesso e colaboração.

Contêiner

Um Dockerfile para rodar um banco de dados PostgreSQL pode ser surpreendentemente simples, pois frequentemente você pode utilizar imagens oficiais disponíveis no Docker Hub. Aqui está um exemplo de um Dockerfile típico para rodar um banco de dados PostgreSQL:

```
# Use a imagem oficial do PostgreSQL como base
FROM postgres:latest

# Defina variáveis de ambiente para o PostgreSQL
ENV POSTGRES_DB=meubanco
ENV POSTGRES_USER=meuusuario
ENV POSTGRES_PASSWORD=minhasenha

# O Dockerfile não precisa de mais instruções porque a imagem do PostgreSQL
# já contém tudo que é necessário para executar o PostgreSQL.
```

Explicação das Instruções

- **FROM postgres:latest:** Este comando define a imagem base como a imagem oficial do PostgreSQL na sua versão mais recente. Você pode especificar uma versão específica substituindo `latest` pela tag da versão desejada.

- **ENV:** Aqui, são definidas três variáveis de ambiente:

    - `POSTGRES_DB`: o nome do banco de dados que será criado automaticamente.

- `POSTGRES_USER`: o nome do usuário com permissões para acessar este banco de dados.
- `POSTGRES_PASSWORD`: a senha para o usuário especificado.

Considerações Adicionais

- **Segurança:** Em um cenário de produção, é aconselhável não codificar a senha diretamente no Dockerfile. Em vez disso, você pode usar mecanismos como variáveis de ambiente definidas externamente, Docker Secrets, ou outras ferramentas de gerenciamento de segredos.
- **Volumes:** Para persistência de dados, você deve considerar o uso de volumes Docker. Isso garante que os dados não sejam perdidos quando o contêiner é destruído e recriado.
- **Porta:** Por padrão, o PostgreSQL escuta na porta 5432. Você pode mapear esta porta para uma porta do host durante a execução do contêiner.

Executando o Contêiner PostgreSQL

Após criar o Dockerfile, você constrói a imagem e executa o contêiner com comandos como:

```
docker build -t meu-postgres .
docker run -d -p 5432:5432 meu-postgres
```

O primeiro comando constrói a imagem a partir do Dockerfile no diretório atual (`.`) e o segundo comando executa o contêiner em modo desanexado (`-d`), mapeando a porta 5432 do contêiner para a porta 5432 do host.

Execute esses comandos com o Dockerfile que passei

Resultados:

```
01-01-intro-docker$ docker build -t meu-postgres .
[+] Building 17.9s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 396B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/postgres:latest
2.1s
 => [1/1] FROM
docker.io/library/postgres:latest@sha256:a80d0c1b119cf3d6bab27f72782f16e47a
b8534ced937fa813ec2ab26e1fd81e
15.7s
 => => resolve
docker.io/library/postgres:latest@sha256:a80d0c1b119cf3d6bab27f72782f16e47a
b8534ced937fa813ec2ab26e1fd81e
0.0s
```

```
 => =>
sha256:8a9a8dd839ec79444b034cd5b9cdf8b67684bdaa4f3706bc66aec2c1c6576bb8
1.16kB / 1.16kB
0.2s
 => =>
sha256:fbd1be2cbb1f55a77e9e4124c36f4f7e7d9370de26d4325688cc953008984c81
9.79kB / 9.79kB
0.0s
 => =>
sha256:9a28d48e5d8f47154e1964a34f986419c1f20ce51b64fa0e610563fc87cd3960
4.42MB / 4.42MB
1.4s
 => =>
sha256:a80d0c1b119cf3d6bab27f72782f16e47ab8534ced937fa813ec2ab26e1fd81e
7.58kB / 7.58kB
0.0s
 => =>
sha256:3648b6c2ac30de803a598afbaaef47851a6ee1795d74b4a5dcc09a22513b15c9
3.13kB / 3.13kB
0.0s
 => =>
sha256:578acb154839e9d0034432e8f53756d6f53ba62cf8c7ea5218a2476bf5b58fc9
29.15MB / 29.15MB
5.1s
 => =>
sha256:a98e37f54ea5ab70d5473f29f3d5ff3a0b1d495930cb836e7fcdd6fd41503a19
1.45MB / 1.45MB
0.9s
 => =>
sha256:7d200efb52088f17b19907b1c97a57a0f61552eb6d0b7bd1e8f617fac6326dda
8.07MB / 8.07MB
2.0s
 => =>
sha256:fe0ac8d1ec65f413b98c7247e330db7c0b12bd599d52438736c1f33c71d2fdaa
1.20MB / 1.20MB
2.1s
 => =>
sha256:85bf04d4b2a20698f920e7ad154fb70ca4c5c7d6b4ba6b91d903ba97e20d1fa1
116B / 116B
2.2s
 => =>
sha256:b96ddf21ebd6cdc8bb3447c91f96f1b49ef2782fb9cd97946b4289fc1bb64b0b
3.14kB / 3.14kB
2.3s
 => =>
sha256:b2983ba85293539382ab00a70bd971f6c5498ed9d772bec382ef18e448e5e4fa
106.39MB / 106.39MB
13.2s
 => =>
sha256:3cbe242a7b19695d29abe23ac258d5f63ccf24d1490a43a5a6f29a626c476e05
9.93kB / 9.93kB
2.6s
 => =>
sha256:0a1ff521257545d79a15659beca23527a388dc6512e4b8540a469f9b1d90ccf0
```

```
128B / 128B
2.8s
 => =>
sha256:f560313d0d85eab368c3644cc6dbb0a5141117b77696a9ae79bd1e4362ae9801
170B / 170B
3.0s
 => =>
sha256:fd463ca121fb5e771b9b486db4438dd9a28713e9bf2f5a8a3bc0f0f248e411e3
4.79kB / 4.79kB
3.2s
 => => extracting
sha256:578acb154839e9d0034432e8f53756d6f53ba62cf8c7ea5218a2476bf5b58fc9
0.9s
 => => extracting
sha256:8a9a8dd839ec79444b034cd5b9cdf8b67684bdaa4f3706bc66aec2c1c6576bb8
0.0s
 => => extracting
sha256:9a28d48e5d8f47154e1964a34f986419c1f20ce51b64fa0e610563fc87cd3960
0.1s
 => => extracting
sha256:a98e37f54ea5ab70d5473f29f3d5ff3a0b1d495930cb836e7fcdd6fd41503a19
0.0s
 => => extracting
sha256:7d200efb52088f17b19907b1c97a57a0f61552eb6d0b7bd1e8f617fac6326dda
0.2s
 => => extracting
sha256:fe0ac8d1ec65f413b98c7247e330db7c0b12bd599d52438736c1f33c71d2fdaa
0.0s
 => => extracting
sha256:85bf04d4b2a20698f920e7ad154fb70ca4c5c7d6b4ba6b91d903ba97e20d1fa1
0.0s
 => => extracting
sha256:b96ddf21ebd6cdc8bb3447c91f96f1b49ef2782fb9cd97946b4289fc1bb64b0b
0.0s
 => => extracting
sha256:b2983ba85293539382ab00a70bd971f6c5498ed9d772bec382ef18e448e5e4fa
2.2s
 => => extracting
sha256:3cbe242a7b19695d29abe23ac258d5f63ccf24d1490a43a5a6f29a626c476e05
0.0s
 => => extracting
sha256:0a1ff521257545d79a15659beca23527a388dc6512e4b8540a469f9b1d90ccf0
0.0s
 => => extracting
sha256:f560313d0d85eab368c3644cc6dbb0a5141117b77696a9ae79bd1e4362ae9801
0.0s
 => => extracting
sha256:fd463ca121fb5e771b9b486db4438dd9a28713e9bf2f5a8a3bc0f0f248e411e3
0.0s
 => exporting to image
0.0s
 => => exporting layers
0.0s
 => => writing image
```

```
sha256:26a3ec18e571dbeefca1ed4ee074d65bda3d546f90004bed9dee19576b29922e
0.0s
 => => naming to docker.io/library/meu-postgres:latest
0.0s

01-01-intro-docker$ docker run -d -p 5432:5432 meu-postgres
f686e04b1f79b574a53fcd8390236cdc93f66c9205a43e5fa4e46be8419cfb02

01-01-intro-docker$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS          PORTS                                     NAMES
f686e04b1f79   meu-postgres   "docker-entrypoint.s…"   11 seconds ago   Up
10 seconds   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   reverent_albattani
01-01-intro-docker$
```

Conseguiu? O primeiro comando é:

```
docker build -t meu-postgres .
```

Este comando "compila" o Dockerfile e cria uma imagem chamada "meu-postgres" na pasta de imagens do nosso Docker local. Note o "." no final do comando, indicando que o Docker file está na pasta atual (esta pasta é o "contexto do Dockerfile").

O cliente Docker vai criar vários contêiners durante o processo de build (um para cada comando) e vai gerar a imagem. Para criarmos um contêiner baseado na imagem que acabamos de compilar, usamos o comando:

```
docker run -d -p 5432:5432 meu-postgres
```

Ele manda executar a imagem "meu-postgres", em modo "detachado" (-d), ou seja, sem prender o terminal, expondo a porta do Contêiner 5432 como porta 5432 do seu computador.

Como sabemos se funcionou? Podemos ver quais contêineres estão ativos no Docker engine com o comando:

```
docker ps
```

A resposta deve ser algo assim:

```
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS          PORTS                                     NAMES
f686e04b1f79   meu-postgres   "docker-entrypoint.s…"   11 seconds ago   Up
10 seconds   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   reverent_albattani
```

O campo **CONTAINER ID** serve para identificarmos o contêiner em comandos futuros. Ele também pode ter um nome (neste caso, não fornecemos o nome).

Apague o que criou

É uma boa prática você sempre remover o que criou, se foi apenas para teste e exercício. Isso poupa custos futuros, quando você estiver utilizando **cloud** services. Para apagar os contêineres criados:

```
docker rm -f <id do contêiner>
```

Para apagar **TODOS** os contêineres (cuidado com esse comando):

```
docker rm -f $(docker ps -q -a)
```

Remova a imagem que criou:

```
docker rmi meu-postgres:latest
```

Se quiser remover **TODAS** as imagens (cuidado com esse comando):

```
docker rmi $(docker images -q)
```