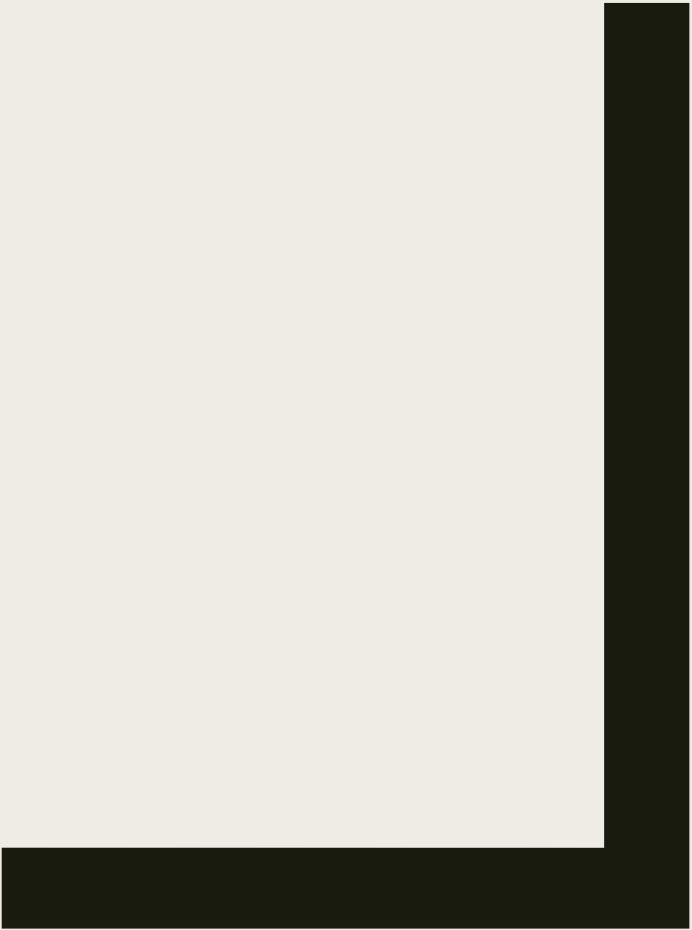




DOCKER

Para Desenvolvedores



Agenda

- O que é Docker
- Porque usar Docker
- Instalação & Execução
- Criando Imagens
- Trabalhando com Volumes
- Ambiente de Desenvolvimento
- Referências
- Perguntas e Respostas

Objetivo

- Apresentar os principais conceitos de Docker:
 - *Principais características*
 - *Funcionamento*
 - *Caso de Uso e Padrões de Projeto*
- Principais comandos
 - *Gerenciamento de Containers e Imagens*
 - *Gerenciamento de Volumes*
- Setup básico para iniciar seu Desenvolvimento
 - *docker-compose*
 - *Flask com MySQL*
- Referências para que vocês possam continuar seus estudos

O que é Docker

- Isolamento de Recursos
 - *Através de Containers*
- Isolamento “com pouca sobrecarga”
 - *Utiliza o mesmo Kernel do Hospedeiro (Host)*
 - Não há emulação de dispositivos de HW
 - Não há necessidade de suporte de HW (ex.: suporte a virtualização no processador)

O que é Docker

■ Máquina Virtual (MV) Vs Containers

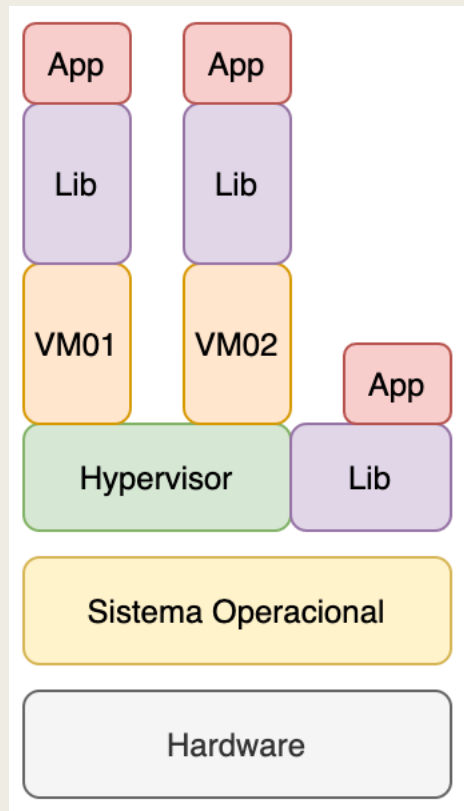
– *Máquina Virtual*

- Hypervisor (Tipo I ou Tipo II)
- Maior isolamento (cada VM com um Kernel/SO próprio)
- Maior sobrecarga

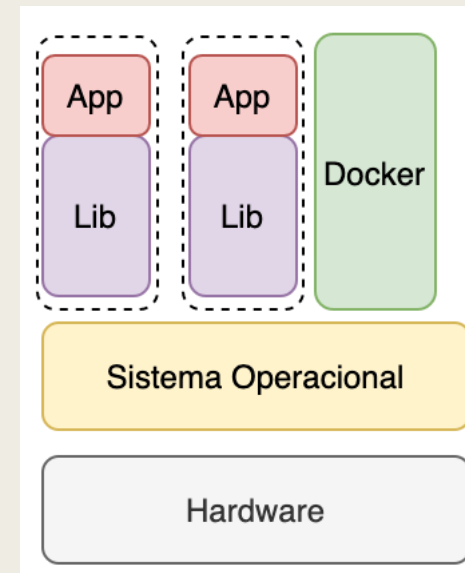
– *Container*

- Não tem o papel de um Hypervisor
- Menor isolamento (compartilhamento de partes do Kernel do hospedeiro)
- Menor sobrecarga

O que é Docker



(a) Máquinas Virtuais



(b) Docker / Containers

O que é Docker

- Containers são baseados no conceito de ***Namespaces***¹ para implementar o isolamento
 - *Feature do Kernel do Linux (desde a versão 2.4, 2002)*
 - *Restringe os recursos que os processos podem “ver” e, portanto, acessar.*
- Containers possuem um ***conjunto de namespaces*** que limitam a “visão” do container para determinados recursos
- Um container possui os seguintes *namespaces*:
 - *PID Namespace: Isolamento de processos*
 - *NET Namespace: Isolamento de interfaces de rede*
 - *IPC Namespace: Isolamento a mecanismos de IPC*
 - *MNT Namespace: Isolamento de file systems (pontos de montagem)*
 - *UTS Namespace: Isolamento de kernel e identificadores de versão*

¹ [Linux containers in 500 lines of code](#)

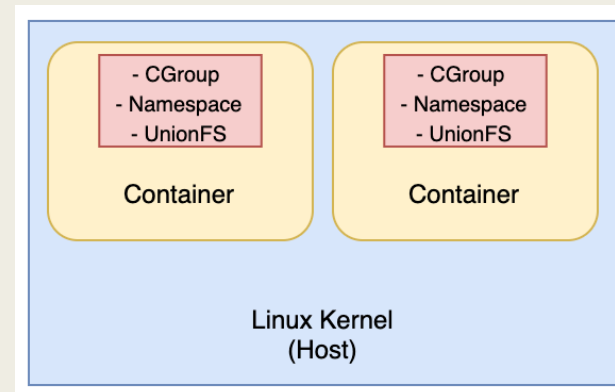
O que é Docker

■ CGroups

- *Mecanismo de alocação de recursos como Tempo de CPU, largura de banda, alocação de memória, etc.*

■ UnionFS

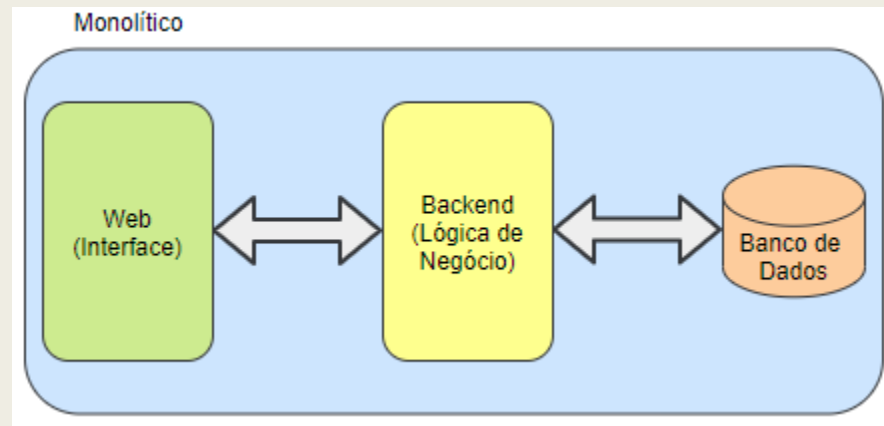
- *Permite que diferentes sistemas de arquivos operem juntos em forma de camadas*
- *Os arquivos nos diferentes FS são “fundidos” (merged)*
- *O mesmo diretório, em diferentes discos, aparece como um diretório maior, pois lista todo o conteúdo dos dois discos*



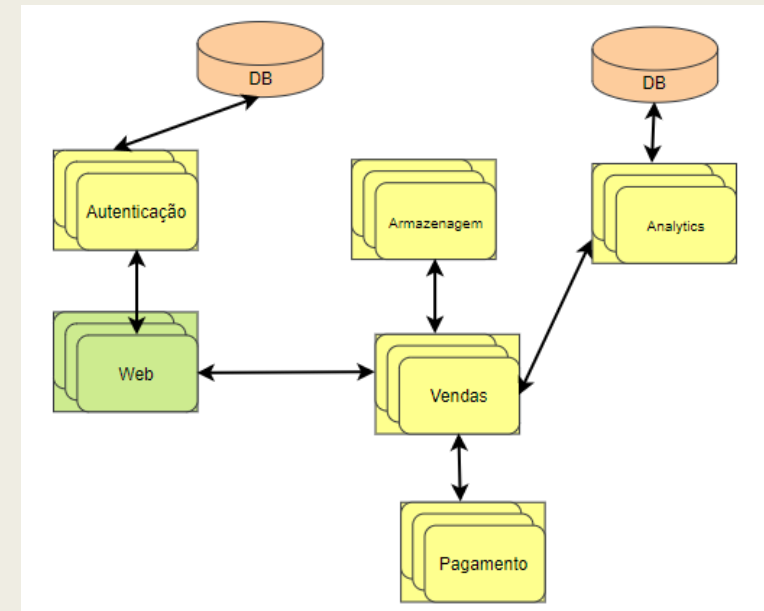
Composição dos Containers

Porque Usar Docker

■ Arquitetura Monolítica Vs Microserviços



(a) Monolítico

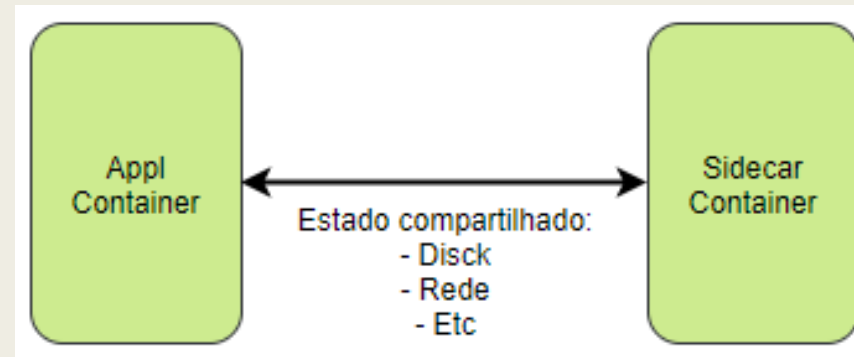


(b) Microserviços

Porque Usar Docker

■ Padrões de Projeto:

- **Sidecar:** Aumentar ou melhorar a aplicação adicionando novas funcionalidades sem a necessidade de alterar a aplicação (ex.: adicionar serviço HTTPS na aplicação, monitoração de serviços)

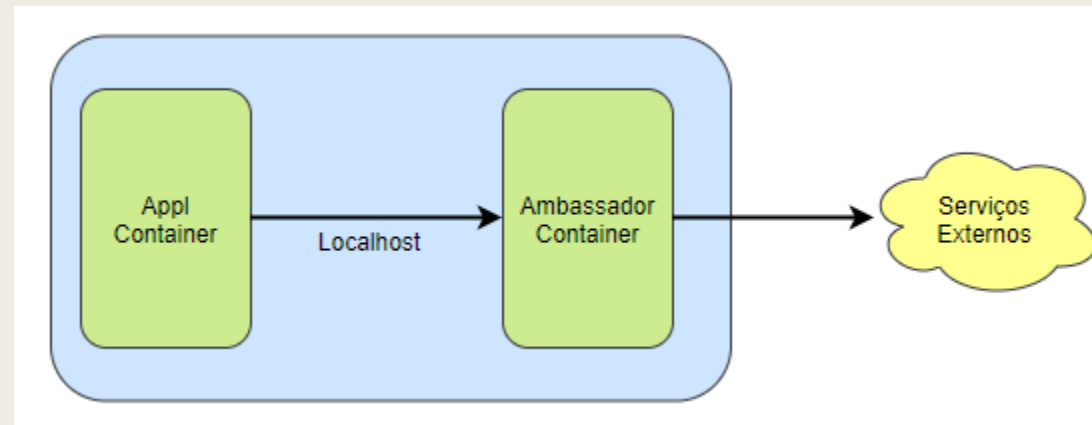


(a) Padrão Sidecar

Porque Usar Docker

■ Padrões de Projeto:

- **Ambassador:** *Faz um papel de proxy nas iterações entre a aplicação e o mundo exterior (Ex.: Sharding, Acesso a um BD, ...)*



(a) Padrão Ambassador

Instalação & Execução

- Instalação em outros SO:
 - Windows: <https://docs.docker.com/docker-for-windows/install/>
 - MacOS: <https://docs.docker.com/docker-for-mac/install/>
- Linux: <https://docs.docker.com/engine/install/>
 - *Adicionar o repositório oficial do docker*
 - *Atualizar cache do apt-get*
 - *Instalar o docker engine*

Instalação & Execução

```
$ sudo docker run ubuntu /bin/echo Hello, Docker!  
Unable to find image 'ubuntu' locally  
Pulling repository ubuntu  
c4ff7513909d: Download complete  
511136ea3c5a: Download complete  
1c9383292a8f: Download complete  
9942dd43ff21: Download complete  
d92c3c92fa73: Download complete  
0ea0d582fd90: Download complete  
...  
cc58e55aa5a5: Download complete  
Hello, Docker!
```

Instalação & Execução

■ O que aconteceu:

- *Docker fez o download de uma imagem base (pois ele não encontrou localmente)*
- *Instanciou um novo container*
- *Configurou uma Interface de Rede e colocou um IP dinâmico*
- *Executou o comando `/bin/echo` (por conta do comando `run`)*
- *Docker capturou a saída do comando e em seguida o docker é terminado*

Instalação & Execução

- Sub-comandos do docker
 - *Verificar o status dos containers*
 - *Verificar imagens instaladas*
 - *Iniciar, Parar e Destruir containers*
 - ...

Instalação & Execução

■ *Top 15 Comandos*

- `docker --version`
- `docker pull <nome_da_imagem>`
- `docker push <nome_usuario/nome_da_imagem>`
- `docker run -it -d <nome_da_imagem>`
- `docker ps`
- `docker ps -a`
- `docker exec -t <id_container> bash`
- `docker stop <id_container>`
- `docker kill <id_container>`
- `docker commit <id_container> <nome_usuario/nome_imagem>`
- `docker login`
- `docker images`
- `docker rm <id_container>`
- `docker rmi <id_imagem>`
- `docker build <path_docker_file>`

Instalação & Execução

```
$ sudo docker run -it ubuntu /bin/bash
root@bc5fdb751dd1:/#

root@bc5fdb751dd1:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.1 LTS"
root@bc5fdb751dd1:/#

root@bc5fdb751dd1:/# ifconfig eth0 | grep 'inet addr:'
inet addr:172.17.0.4 Bcast:0.0.0.0 Mask:255.255.0.0
root@bc5fdb751dd1:/# exit
```

Instalação & Execução

```
$ sudo docker ps -a
CONTAINER ID = bc5fdb751dd1
IMAGE = ubuntu:latest
COMMAND = "/bin/bash"
CREATED = 13 minutes ago
STATUS = Exited (0) 4 minutes ago
PORTS
NAMES = happy_fermi

$ sudo docker ps -qa
bc5fdb751dd1

$ sudo docker stats bc5fdb751dd1
CONTAINER    CPU % MEM    USAGE/LIMIT    MEM % NET I/O
bc5fdb751dd1 0.00% 5.098 MiB/7.686 GiB 0.06% 648 B/648 B

$ sudo docker images
REPOSITORY TAG          IMAGE ID      CREATED          VIRTUAL SIZE
ubuntu     latest      eca7633ed783 37 hours ago    192.7 MB
```

Instalação & Execução

```
$ sudo docker ps -qa
bc5fdb751dd1

$ sudo docker rm bc5fdb751dd1
bc5fdb751dd1

$ sudo docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS   NAMES
```

Instalação & Execução

```
$ sudo docker run -it --name ex_nginx ubuntu
root@943b6186d4f8:/#

root@943b6186d4f8:/# apt-get update
root@943b6186d4f8:/# apt-get install -y nginx
root@943b6186d4f8:/# nginx -v
nginx version: nginx/1.4.6 (Ubuntu)
root@943b6186d4f8:/# exit

$ sudo docker commit f637c4df67a1 ubuntu/nginx
78f4a58b0f314f5f81c2e986d61190fbcc01a9e378dcbebf7c6c16786d0a270c

$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           VIRTUAL SIZE
ubuntu/nginx        latest       78f4a58b0f31     55 seconds ago   231.2 MB
ubuntu              latest      eca7633ed783     38 hours ago     192.7 MB

$ sudo docker run -it --rm -p 8080:80 ubuntu/nginx /bin/bash
root@e98f73a63965:/# service nginx start
```

Instalação & Execução

- <http://localhost:8080>
 - `-p <porta_local>:<porta_container>`
 - `-p <porta_local>:<porta_container>/udp`



Instalação & Execução

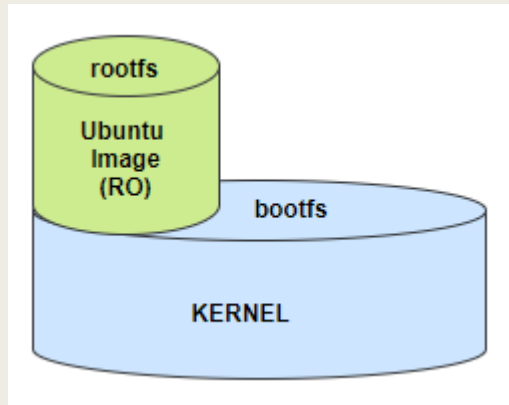
```
$ sudo docker run -d -p 8080:80 ubuntu/nginx /usr/sbin/nginx -g "daemon off;"  
574911d47ca2fc8da14e59843b0e319ca8899215f7ee3b9a1a41f708a72b7495
```

Criando Imagens

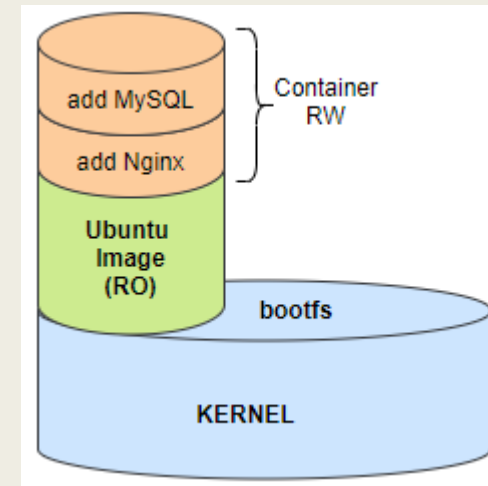
■ Processo de criação de uma Imagem

- *Docker monta o rootfs em modo read-only (similar ao processo tradicional de boot do Linux)*
- *O rootfs contém uma estrutura típica de diretórios (i.e. /dev, /proc, /etc, /lib..)*
- *Entretanto.... O docker não muda o sistema de arquivos para read-write, utiliza o unionfs, isto é, cria uma camada read-write ACIMA da camada read-only*

Criando Imagens



(a) Camadas durante a criação do Container



(b) Camadas RO e RW do container após a criação do container

Criando Imagens

■ Dockerfile

- *Automatização de tarefas no Docker*
- *Descrição do que deve ser inserido na Imagem*
- *É criado um Snapshot com a instalação que elaboramos no Dockerfile*

Criando Imagens

```
FROM ubuntu
MAINTAINER Artur Baruchi <abaruchi@abaruchi.dev>
RUN apt-get update
RUN apt-get install -y nginx
```

- FROM: Imagem Base
- MAINTAINER: Nome de quem vai manter a imagem
- RUN: Executa comandos no container

```
$ sudo docker build -t nginx .
```

Criando Imagens

```
FROM ubuntu
MAINTAINER Daniel Romero <infoslack@gmail.com>
RUN apt-get update
RUN apt-get install -y nginx
EXPOSE 80
```

```
$ sudo docker build -t nginx .
Sending build context to Docker daemon 2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu
---> eca7633ed783
Step 1 : MAINTAINER Daniel Romero <infoslack@gmail.com>
---> f6fe0fa267d2
Step 2 : RUN apt-get update
---> Using cache
---> 2ed51e9c4e64
Step 3 : RUN apt-get install -y nginx
---> Using cache
---> 37bb1a8d7ea6
Step 4 : EXPOSE 80
---> Running in 94b7bd18ac37
---> 3568a7119034
Removing intermediate container 94b7bd18ac37
Successfully built 3568a7119034
```

Criando Imagens

```
FROM ubuntu
MAINTAINER Daniel Romero <infoslack@gmail.com>
RUN apt-get update
RUN apt-get install -y nginx
ADD exemplo /etc/nginx/sites-enabled/default
EXPOSE 80
```

- O arquivo chamado `exemplo` será copiado para `/etc/nginx/sites-enabled/` com o nome `default`;
- O arquivo `exemplo` deverá existir no mesmo diretório do Dockerfile;

```
server {
    listen 8080 default_server;
    server_name localhost;

    root /usr/share/nginx/html;
    index index.html index.htm;
}
```

Criando Imagens

```
$ sudo docker build -t nginx .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu
---> eca7633ed783
Step 1 : MAINTAINER Daniel Romero <infoslack@gmail.com>
---> Using cache
---> f6fe0fa267d2
Step 2 : RUN apt-get update
---> Using cache
---> 2ed51e9c4e64
Step 3 : RUN apt-get install -y nginx
---> Using cache
---> 37bb1a8d7ea6
Step 4 : ADD exemplo /etc/nginx/sites-enabled/default
---> 3eb7bb07b396
Removing intermediate container 3d5b9c346517
Step 5 : EXPOSE 8080
---> Running in de3478bbb1ff
---> f12246bf988d
Removing intermediate container de3478bbb1ff
Successfully built f12246bf988d
```

Criando Imagens

- Podemos migrar a imagem criada para outro servidor usando o comando `save` e depois o `load`
- Interessante para projetos privados ou para usar em times pequenos de desenvolvimento.

```
$ sudo docker ps -q
b02af9430141

$ sudo docker commit b02af9430141 nova_imagem
9a8c0a1a72d2f9c2815e455a22be91f9cf788f769d2cca5b603baebd10ccc38a

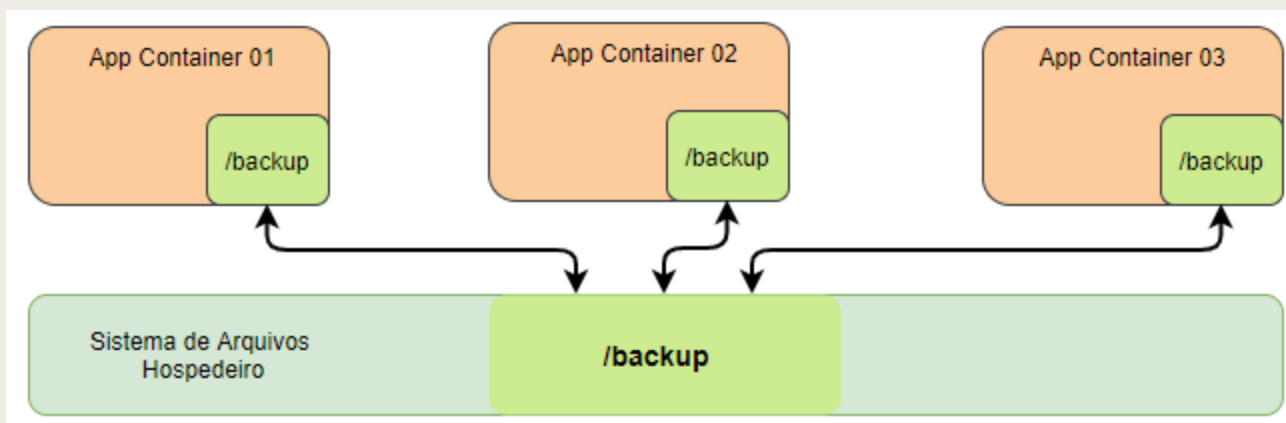
$ sudo docker save nova_imagem > /tmp/nova_imagem.tar

$ sudo docker load < /tmp/nova_imagem.tar
```

Trabalhando com Volumes

■ Volume

- *Pode ser um diretório fora do sistema de arquivos de um container*
- *Podemos mapear um diretório no sistema de arquivos local para dentro do container*
- *Mesmo volume pode ser mapeado entre diversos containers*



Volume compartilhado entre os containers.

Trabalhando com Volumes

```
$ sudo docker run -d -p 8080:8080 -v /tmp/nginx:/usr/share/nginx/html:ro nginx
c88120d5efbeb0607ea861ad9f0d9a141eb4ecb18a518a096f9ad38ebe9207bd

$ curl -IL http://localhost:8080
HTTP/1.1 403 Forbidden
Server: nginx/1.4.6 (Ubuntu)

$ echo "It works!" > /tmp/nginx/index.html
$ curl http://localhost:8080
It works!
```


Trabalhando com Volumes

```
FROM ubuntu
MAINTAINER Artur Baruchi <abaruchi@abaruchi.dev>
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update -qq && apt-get install -y mysql-server-5.5
ADD my.cnf /etc/mysql/conf.d/my.cnf
RUN chmod 664 /etc/mysql/conf.d/my.cnf
ADD run /usr/local/bin/run
RUN chmod +x /usr/local/bin/run
VOLUME ["/var/lib/mysql"]
EXPOSE 3306
```

- ENV: Declaração de Variável de Ambiente (nesse caso estamos evitando que durante a instalação do MySQL alguma tela interativa)
- VOLUME: Informando que o diretório `/var/lib/mysql` do Container será um volume, que será criado dentro do diretório `/var/lib/docker/volumes/<uuid>`
- EXPOSE: Indica que a porta deverá ser exposta para que seja acessível da máquina hospedeira

Trabalhando com Volumes

- Comando específico para o gerenciamento de Volumes

```
$ docker volume ls
DRIVER          VOLUME NAME
local           26f2a779cb970781975a1e1f571fe223c8816108aa77e4bbd4e74099cb237a5c
local           53d10271fa1cccca264ad16413d8ac0720cd5b0282eee08ed23f17929db2209
local           206d50a08d359013094b61e40301d6ce2fa5fdc6123f46f6f99603288f5b740c

$ docker volume inspect 26f2a779cb970781975a1e1f571fe223c8816108aa77e4bbd4e74099cb237a5c
[
  {
    "CreatedAt": "2019-02-28T19:13:11Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/26f2a779cb970781975a1e1f571fe223c8816108aa77e4bbd4e74099cb237a5c/_data",
    "Name": "26f2a779cb970781975a1e1f571fe223c8816108aa77e4bbd4e74099cb237a5c",
    "Options": null,
    "Scope": "local"
  }
]
```

Trabalhando com Volumes

```
$ docker volume create my_data  
my_data
```

```
$ docker volume ls
```

DRIVER	VOLUME NAME
local	8e0a283dcb085a31eea7dd370c5c89be15c850148c5da0c5f8cdfb5b89cdb1c4
local	26f2a779cb970781975a1e1f571fe223c8816108aa77e4bbd4e74099cb237a5c
local	53d10271fa1ccccda264ad16413d8ac0720cd5b0282eee08ed23f17929db2209
local	my_data

Ambiente de Desenvolvimento

- Utilizar o *Docker Compose*
 - *Automatização de tarefas*
 - *Reduz a complexidade na configuração do Ambiente*
- Aplicação *Flask com MySQL*

Ambiente de Desenvolvimento

- Instalar o docker-compose

```
$ sudo pip install -U docker-compose
```

- O docker-compose consiste em 3 passos:
 - *Definição do Dockerfile da Aplicação;*
 - *Definição dos serviços;*
 - *Criação do serviço;*

Ambiente de Desenvolvimento

■ Criar uma Estrutura de diretórios para a aplicação

```
$ cd $HOME
$ mkdir -p flask_app/app flask_app/db
$ cd flask_app
```

```
flask_app
|
| - docker-compose.yml
|
| - app/
|   |
|   | - Dockerfile
|   | - app.py
|   | - requirements.txt
|
| - db/
|   |
|   | - init.sql
```

Ambiente de Desenvolvimento

- Criar o Dockerfile da nossa Aplicação (Flask)

```
FROM python:3.6

EXPOSE 5000

WORKDIR /app

COPY requirements.txt /app
RUN pip install -r requirements.txt

COPY app.py /app
CMD python app.py
```

Ambiente de Desenvolvimento

- Criar o arquivo `app.py` (que é onde tem o nosso código principal)

```
from typing import List, Dict
from flask import Flask
import mysql.connector
import json

app = Flask(__name__)
def favorite_colors() -> List[Dict]:
    config = {
        'user': 'root',
        'password': 'root',
        'host': 'db',
        'port': '3306',
        'database': 'knights'
    }
    connection = mysql.connector.connect(**config)
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM favorite_colors')
    results = [{name: color} for (name, color) in cursor]
    cursor.close()
    connection.close()

    return results

@app.route('/')
def index() -> str:
    return json.dumps({'favorite_colors': favorite_colors()})

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```


Ambiente de Desenvolvimento

- Criar o arquivo `requirements.txt`

```
Flask  
mysql-connector
```

Ambiente de Desenvolvimento

- Criar o arquivo `init.sql`

```
CREATE DATABASE knights;
use knights;

CREATE TABLE favorite_colors (
  name VARCHAR(20),
  color VARCHAR(10)
);

INSERT INTO favorite_colors
(name, color)
VALUES
('Lancelot', 'blue'),
('Galahad', 'yellow');
```

Ambiente de Desenvolvimento

■ Criar o arquivo docker-compose.yml

```
version: "2"
services:
  app:
    build: ./app
    links:
      - db
    ports:
      - "5000:5000"

  db:
    image: mysql:5.7
    ports:
      - "32000:3306"
    environment:
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - ./db:/docker-entrypoint-initdb.d/:ro
```

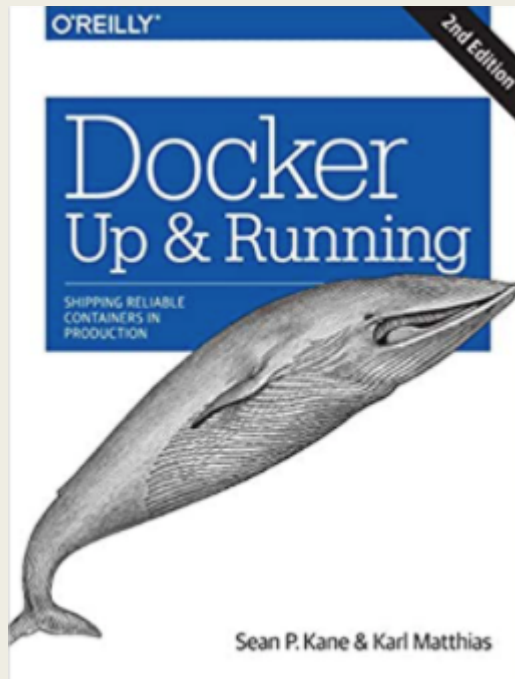
Ambiente de Desenvolvimento

■ Subimos o serviço

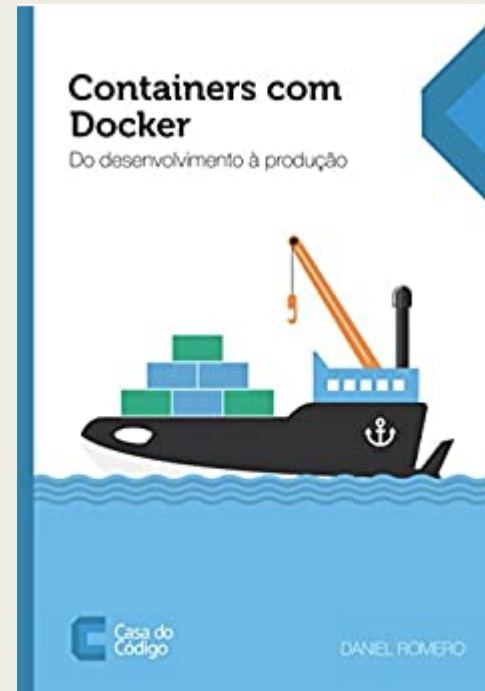
```
$ sudo docker-compose up  
  
...  
  
app_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Referências

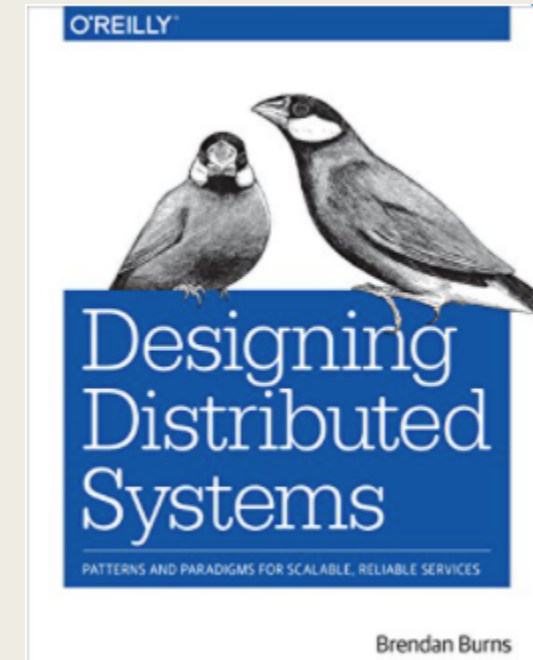
■ Livros



■ Docker: Up and Running



■ Containers com Docker Desenvolvimento à Produção



■ Designing Distributed Systems

Referências

- Sites

- *Django Cookiecutter*

- <https://realpython.com/development-and-deployment-of-cookiecutter-django-via-docker/>

- *Repositório Docker Cookbook*

- <https://github.com/PacktPublishing/Docker-Cookbook-Second-Edition>

Perguntas & Respostas

- Twitter:
 - [@abaruchi](https://twitter.com/abaruchi)
- Website:
 - <https://abaruchi.dev>
- Blog:
 - <https://blog.abaruchi.dev>
- Email:
 - abaruchi@abaruchi.dev