docker

DEV.SAJ ADV

André Justi

O que é **Docker**

O Docker é uma tecnologia Open Source que permite criar, executar, testar e implantar aplicações distribuídas dentro de containers de software. Ele permite que você empacote um software de uma padronizada para o desenvolvimento de software, contendo tudo que é necessário para a execução: código, runtime, ferramentas, bibliotecas, etc. O Docker permite que você implante aplicações rapidamente, de modo confiável e estável, em qualquer ambiente.

O que são **containers**

Os containers são um método de virtualização em nível de sistema operacional que permite executar uma aplicação e suas dependências em processos com recursos isolados. Os containers permitem empacotar facilmente o código, as configurações e as dependências de uma aplicação em elementos fundamentais que oferecem consistência ambiental, eficiência operacional, produtividade de desenvolvedores e controle de versões.

Os containers podem ajudar a garantir rapidez, confiabilidade e consistência de implantação, independentemente do ambiente de implantação. Além disso, os containers oferecem um controle mais granular dos recursos, aumentando a eficiência da infraestrutura.

Breve **história**

A construção do Docker foi iniciada por Solomon Hykes, na França, dentro da empresa DotCloud, Docker representa a evolução da tecnologia proprietária da DotCloud.

Docker foi lançado como Open Source em março de 2013, e em março de 2014 com o lançamento da versão 0.9 Docker que deixou LXC como ambiente de execução padrão para usar sua própria libcontainer, que é escrita na linguagem GO criada pelo Google.

No dia 24 de outubro de 2015 o projeto tornou-se o 20º mais estrelado, como mais de 6.800 fork's, como mais de 1100 colaboradores. Uma breve análise de 2016 mostrou DotCloud, Cisco, Google, IBM, Microsoft e Red Hat como as principais contribuidoras do Docker.

Alguns outros fatos sobre **Docker**

Existem mais de 500 mil aplicações Dockerizadas, um crescimento de 3100% ao longo de 2 anos.

Mais de 4 bilhões de containers já foram puxados até hoje.

Docker é apoiado por uma grande e crescente comunidade de colaboradores e usuários; Como exemplo, há 150 mil membros de Meetups Docker em todo o mundo. Isso é cerca de 40% da população da Islândia!

A adoção do Docker aumentou mais de 30% no último ano.

Cerca de 30% dos containers Dockers estão rodando em produção.

29% das empresas que já ouviram falar em Docker planejam usá-lo.

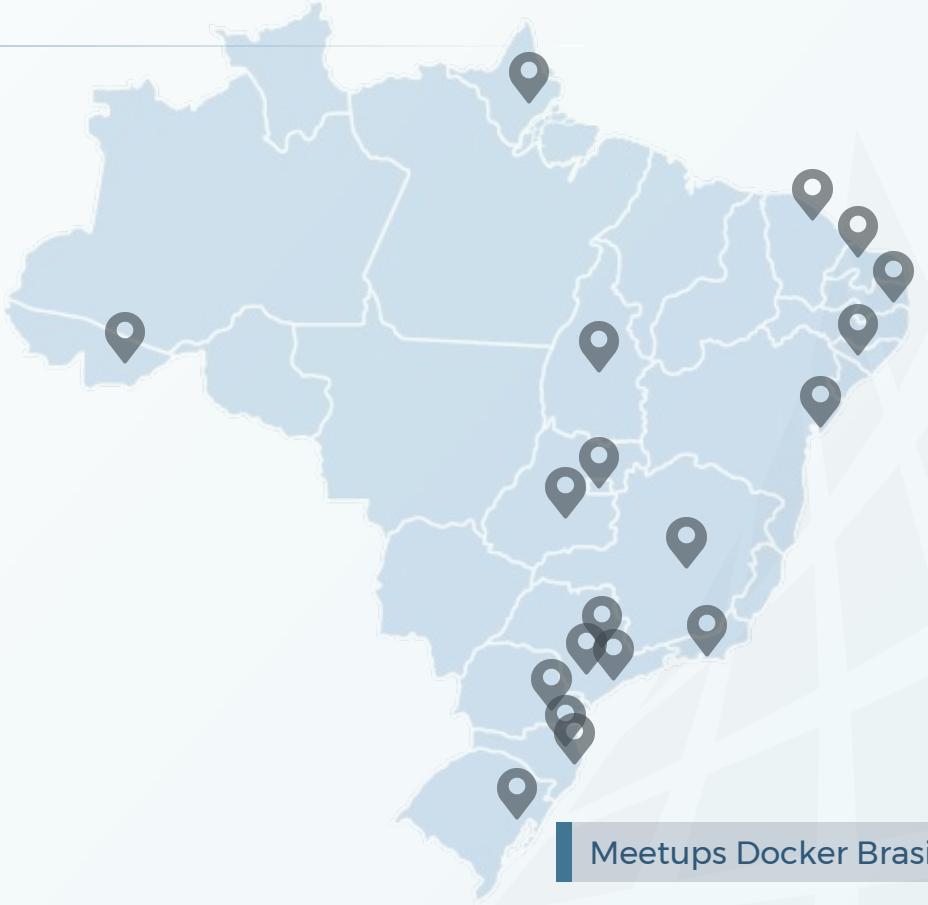
Alguns outros fatos sobre **Docker**

Uma análise de janeiro 2017 dos perfis do LinkedIn, mostra que as skils sobre Docker cresceram 160% em 2016.

No GitHub existem quase 150 mil repositórios com algum código ou artefato Docker.

Comunidade

Com certeza a comunidade Docker é um dos seus pontos fortes, são centenas de grupos de meetups espalhados pelo mundo, além de fóruns, grupos de facebook e milhares de contribuidores no Twitter, GitHub, YouTube, SlideShare etc.



Meetups Docker Brasil

O que estão falando por aí | **Google Trends**

Principais buscas

1. docker
2. container docker
3. container
4. install docker
5. dockerfile
6. ubuntu docker
7. docker image
8. linux docker
9. docker run
10. docker windows
11. hub docker
12. docker file
13. docker compose
14. docker containers
15. docker images

Comparação com outros termos populares

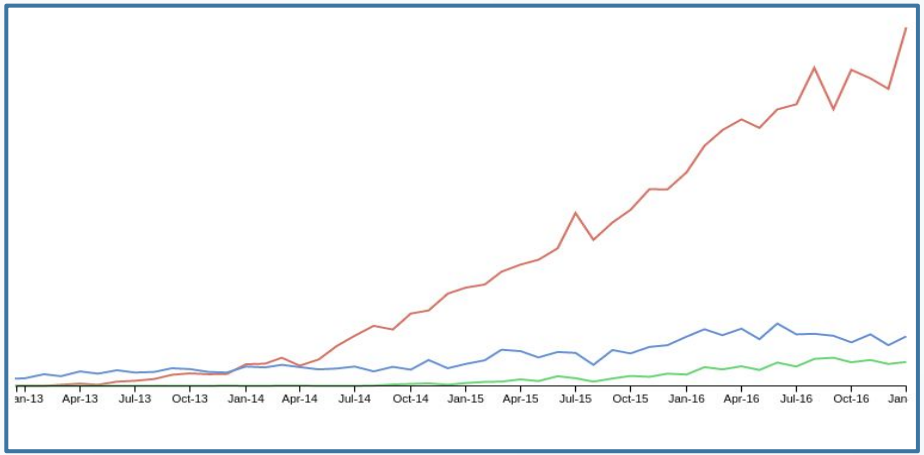
- Docker
- Big Data
- Micro Services

2013 ◄ ····· ► 2017

O que estão falando por aí | **Stack Overflow**

Comparação com outros termos populares

- Docker
- Big Data
- Micro Services



2013 ◄ ····· ► 2017

Quem está usando

... e muitas outras empresas e projetos!

VM vs Docker

VM

O objetivo desse modelo é compartilhar os recursos físicos entre vários ambientes isolados, sendo que cada um deles tem sob sua tutela uma máquina inteira, com memória, disco, processador, rede e outros periféricos, todos entregues via abstração de virtualização.

É como se dentro de uma máquina física criasse máquinas menores e independentes entre sí. Cada máquina dessa tem seu próprio sistema operacional completo, que por sua vez interage com todos os hardwares virtuais que lhe foi entregue pelo modelo de virtualização a nível de máquina.

Vale ressaltar que o sistema operacional instalado dentro de uma máquina virtual fará interação com os hardwares virtuais e não com o hardware real.

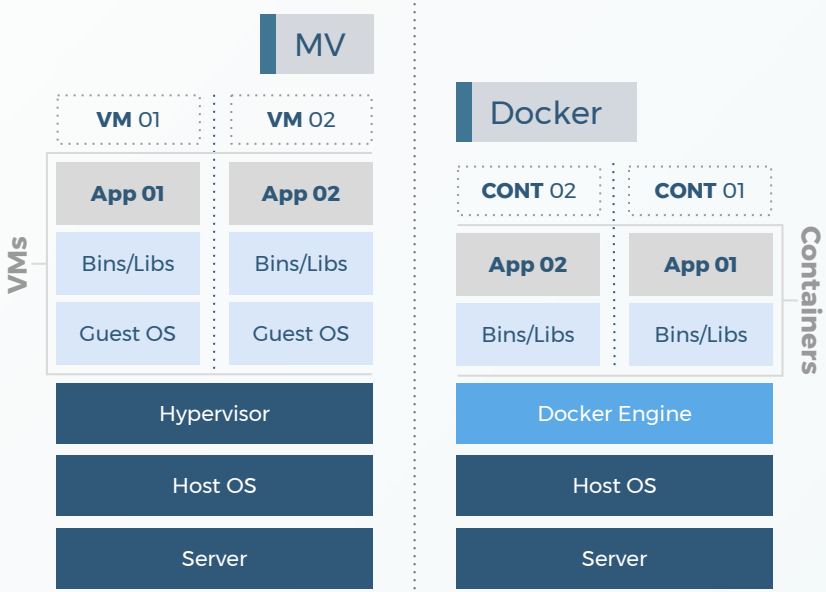
VM vs **Docker**

Docker

Esse modelo de virtualização está no nível de sistema operacional, ou seja, ao contrário da máquina virtual um container não tem visão de uma máquina inteira, ele é apenas um processo em execução em um kernel compartilhado entre todos os outros containers.

Ele utiliza o namespace para prover o devido isolamento de memória RAM, processamento, disco e acesso a rede, mesmo compartilhamento o mesmo kernel, esse processo em execução tem a visão de estar usando um sistema operacional dedicado.

VM vs Docker



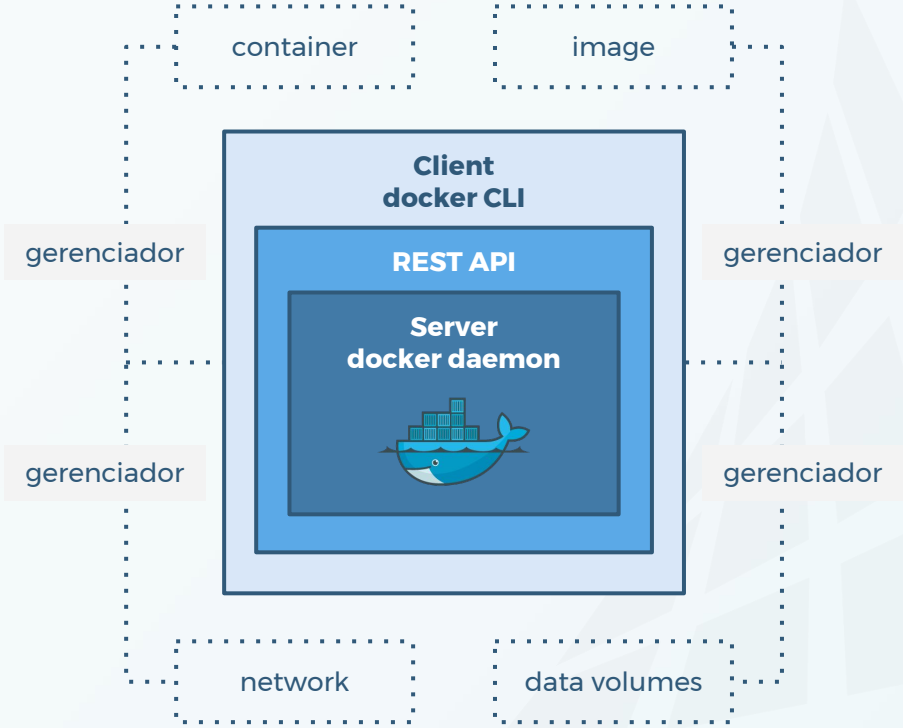
Estrutura **VM** (Virtual Machine)

Estrutura **Docker**

Partes principais do **Docker**

Docker usa uma arquitetura cliente-servidor. A parte cliente fala com o Docker daemon, que faz o trabalho pesado de construção, execução e distribuição de seus containers e imagens Docker, também controla os recursos executados.

O cliente Docker e Docker daemon, podem ser executados no mesmo sistema, também é possível conectar um cliente Docker a um Docker daemon remoto. O cliente Docker e daemon se comunicam através de uma API REST, através de sockets UNIX ou uma interface de rede, para execuções de comandos ou scripts.



Elementos do **Docker**

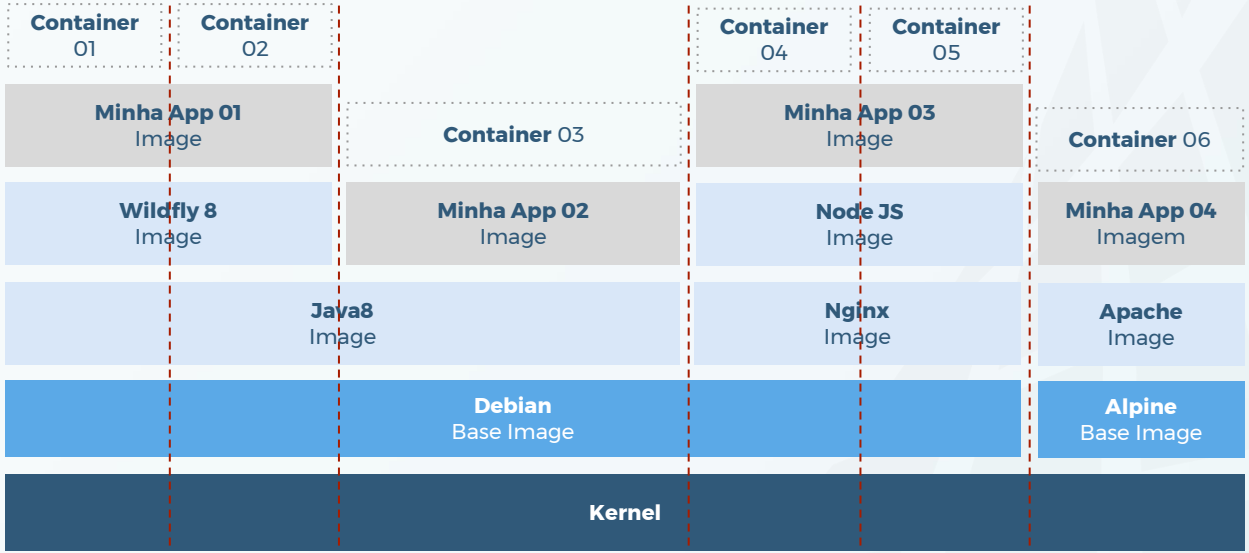
Containers docker - Containers tem como base sempre uma imagem, pense como na seguinte analogia do mundo Java, uma imagem é uma classe e um container é como um objeto instância dessa classe, então podemos através de uma imagem "instanciar" vários containers, também através de recursos chroot, Cgroups é possível definirmos limitações de recursos recursos e isolamento parcial ou total dos mesmos.

Algumas características dos containers

- Portabilidade de aplicação
- Isolamento de processos
- Prevenção de violação externa
- Gerenciamento de consumo de recursos

Elementos do **Docker**

imagens docker - Imagens são templates para criação de containers, como falado no slide anterior, imagens são imutáveis, para executá-las é necessário criar uma instância dela o "container", também vale ressaltar que as imagens são construídas em camadas, o que facilita sua reutilização e manutenção. Em resumo uma imagem nada mais é do que um ambiente totalmente encapsulado e pronto para ser replicado onde desejar



Elementos do **Docker**

Dockerfile - São scripts com uma série de comandos para criação de uma imagem, nesses scripts podemos fazer uma séries de coisas como executar comandos sh, criar variáveis de ambiente, copiar arquivos e pastas do host para dentro da imagem etc.

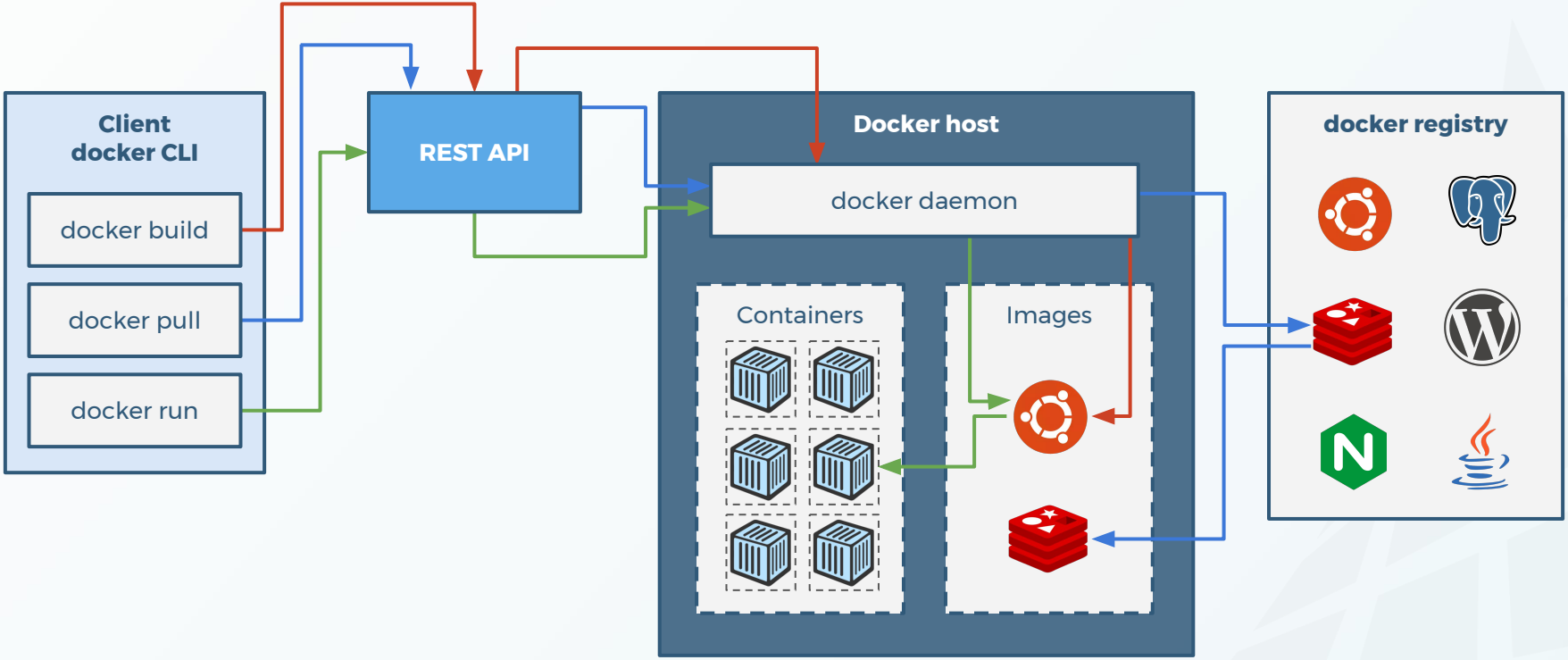
Exemplo

```
FROM ubuntu

MAINTAINER andrejusti

RUN apt-get update
RUN apt-get install -y nginx && apt-get clean
RUN ln -sf /dev/stdout /var/log/nginx/access.log
RUN ln -sf /dev/stderr /var/log/nginx/error.log
RUN echo "daemon off;" >> /etc/nginx/nginx.conf

EXPOSE 8080

ENTRYPOINT ["/usr/sbin/nginx"]

CMD ["start", "-g"]
```

Elementos do **Docker**

Docker Registry - É como um repositório GIT, onde as imagens podem ser versionadas, comitadas, “puxadas” etc, quando recuperamos uma imagem, usando o comando docker pull por exemplo, estamos normalmente baixando a imagem de um registro Docker, o repositório oficial do Docker é o Docker HUB, onde é possível hospedar e versionar imagens públicas e privadas.

Partes principais do **Docker**



Comandos Docker

Tipo de parâmetro	Descrição
<valor>	parâmetro obrigatório
[valor]	parâmetro opcional
[valor...] ou <valor...>	parâmetro multivalorado

Comandos Docker | **docker help**

docker <COMAND> **--help**

Exibe a forma de execução do comando e seus possíveis parâmetros

Exemplo

```
$ docker logs --help
> Usage:    docker logs [OPTIONS] CONTAINER
> Fetch the logs of a container
> Options:
>       --details        Show extra details provided to logs
>   -f, --follow         Follow log output
>       --help           Print usage
>       --since string   Show logs since timestamp
>       --tail string    Number of lines to show from the end of the logs (default ""all"")
>   -t, --timestamps     Show timestamps
```

Comandos Docker | **docker info**

docker info

Exibe as informações de execução do docker

Exemplo

```
$ docker info
> Containers: 5
>  Running: 1
>  Paused: 0
>  Stopped: 4
> Images: 17
> Server Version: 1.13.0
> Storage Driver: overlay2
>  Backing Filesystem: extfs
>  Supports d_type: true
>  Native Overlay Diff: true
> Logging Driver: json-file
> Cgroup Driver: cgroupfs
> Plugins:
>  Volume: local
>  Network: bridge host macvlan null overlay
```

Comandos Docker | **docker login**

docker login [OPTIONS] <SERVER>
Faz login em um servidor de registro Docker

Parâmetros	
-u	Login do registro docker
-p	Senha do registro docker

Exemplo

```
$ docker login -u meuLogin -p minhaSenha docker-registro.meuservidor.com.br
> Login Succeeded

#Exibir servidores logado
$ cat $HOME/.docker/config.json
> "auths": {"docker-registro.meuservidor.com.br": {"auth": "c2FqYWR2OkBTb2Z0cGxhbjIwMTI="}}
```

Comandos Docker | **docker logout**

docker logout <SERVER>
Faz login em um servidor de registro Docker

Exemplo

```
$ docker logout docker-registro.meuservidor.com.br
> Removing login credentials for docker-registro.meuservidor.com.br
```

Comandos Docker | **docker images**

docker images [OPTIONS]
Lista as imagens baixadas

Parâmetros	
-a	Mostrar todas imagens (por padrão oculta as intermediárias)

Exemplo

```
$ docker images
> REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
> redis               latest              74d8f543ac97        6 days ago          184 MB
> ubuntu              latest              f49eec89601e        2 weeks ago         129 MB

$ docker images -a
> REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
> redis               latest              74d8f543ac97        6 days ago          184 MB
> ubuntu              latest              f49eec89601e        2 weeks ago         129 MB
> java                latest              d23bdf5b1b1b        2 weeks ago         643 MB
```

Comandos Docker | **docker pull**

docker pull [OPTIONS] <NAME>[:TAG]
Baixa uma imagem

Parâmetros	
-a	Baixa todas tags da imagem

Exemplo

```
$ docker pull php:latest
> latest: Pulling from library/php
> 5040bd298390: Already exists
> 568dce68541a: Pull complete
> 6a832068e64c: Pull complete
> Digest: sha256:cdd9431e016e974cc84bb103e22152195e02f54591ac48fe705d66b1384d6a08
> Status: Downloaded newer image for php:latest
```

Comandos Docker | **docker search**

docker search [OPTIONS] <TERM>
Pesquisa nos repositórios imagens

Parâmetros	
--limit <quantidadeResultados>	Limita a consulta a um número de resultados

Exemplo

```
$ docker search redis
> NAME                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
> redis                     Redis is an open source key-value store th...   3336      [OK]
> sameersbn/redis                                                           43                   [OK]
> bitnami/redis             Bitnami Redis Docker Image                      36                   [OK]
> torusware/speedus-redis   Always updated official Redis docker image...   32                   [OK]

$ docker search --limit 1 redis
> NAME                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
> redis                     Redis is an open source key-value store th...   3336      [OK]
> sameersbn/redis                                                           43                   [OK]
```

Comandos Docker | **docker run**

docker run [OPTIONS] <IMAGE> <COMMAND> [ARG...]
Executa uma imagem (criando um container)

Parâmetros	
-e	Define uma variável de ambiente
--env-file	Local de uma arquivo com variáveis de ambiente
--link	Adicionar link a outro container
-m <quantidadeDeMemoria>	Define o limite de memória que o container pode usar do host
--memory-swap <quantidadeDeMemoria>	Define o limite de memória swap que o container pode usar (-1 para deixar ilimitado)
--name	Nome do container

Comandos Docker | **docker run**

Parâmetros	
-p	Mapeia uma porta entre o container e o host
--restart	Tipo de política de reinicialização do container **Opções:** **no** - Não reinicie quando o serviço for inicial **failure** - Reinicia somente se o container foi encerrado com status diferente de zero (diferente de sucesso) **always** - Sempre reinicia quando o serviço docker for iniciado, independente do status
--rm	Remover o container automaticamente quando ele for terminado (default false) --runtime Tempo de execução para usar neste container
-v	Vincular um volume
--storage-opt	Opções do storage do container
-d	Roda o container em backgroud

Comandos Docker | **docker run**

Exemplo

```
$ docker run -d -e MINHA_VARIAVEL=minhaVariavel redis:latest
> 840bf6fa81cb026d15fd2c514e25bd5d2b36bea6f98650428adfd786eb559f3a

$ docker run -d  -m 100m --memory-swap 120m redis:latest
> 01bf438ccf92043c3b67bfe06215b05ce6f898dd83d73465279ca3cbd7a97e61

$ docker run --rm -u root redis:latest
> 27b6da614858f24c9757b8a4f27d4d3526809b1592be725ba251b71d23960727

$ docker run -v $HOME/dev/temp:/docker --name meuRedis redis:latest
> 0471a8bd44ef5833ae81a5de8b6863451b5a1bc1047d5a34e7003720a4c34068
```

Comandos Docker | **docker run --link**

LINKS - Comunicação via Link

O Link permite que você trafegue informações entre os containers de forma segura, pois quem conhece um container conhece apenas o seu par definido no link. Quando você configura um link, você cria um elo de ligação entre um container de origem e um container de destino. Para criar um link, você deve utilizar o parâmetro –link no comando docker run. Em primeiro lugar, deve-se criar um container que será origem de dados para outro container.

OBS: Isso também pode ser feito deixando uma porta acessível do container para outro se contactar.

Exemplo

```
# Isso criará um novo container chamado db a partir da imagem do postgres, que contém um banco de dados PostgreSQL.
$ docker run -d --name db training/postgres

# Isso criará um novo container chamado web e irá vincular ao container chamado db.
$ docker run -d -P --name web --link db:db training/webapp python app.py

# Isso irá mostrar que os dois container web tem acesso ao container db.
$ docker exec web ping db
> PING db (172.17.0.2) 56(84) bytes of data.
> 64 bytes from db (172.17.0.2): icmp_seq=1 ttl=64 time=0.098 ms
> 64 bytes from db (172.17.0.2): icmp_seq=2 ttl=64 time=0.224 ms
```

Comandos Docker | **docker run -v**

O volume do Docker é a única maneira de preservar os dados do container em execução, visto que uma vez que o container é removido ele perde todos seus dados, através do volume então é possível, mapear uma pasta no host ou em outro container para uma pasta do container em execução, assim uma vez que um arquivo for criado, atualizado, deletado e etc no container de execução isso será persistido em outro local assim sendo possível preservar esses dados.

Exemplo

```
$ docker run --name redis-test -v $HOME/temp/docker:/data/temp/docker redis
> 36939054e8653242a1be06163e6f6c75420381a3edb4a52b06e62a2844be1914

$ docker exec redis-teste touch /data/temp/docker/arquivo.txt

$ ls $HOME/temp/docker
> arquivo.txt
```

Comandos Docker | **docker inspect**

docker inspect <CONTAINER>
Mostra os metadados do container, como os volumes associados, mantenedor etc.

Exemplo

```
$ docker inspect db
> [
>         "Id": "1045cb2ec8f89b5846bcaca3303dae7b1d6416cffbfc2d314203006221e7c352",
>         "Created": "2017-03-06T18:57:04.851794871Z",
>         "Path": "su",
>         "Args": [
>             "postgres",
>             "-c",
>             "/usr/lib/postgresql/$PG_VERSION/bin/postgres -D /var/lib/postgresql/$PG_VERSION/main/ -c
config_file=/etc/postgresql/$PG_VERSION/main/postgresql.conf"
> # Outras informações
> ]
```

Comandos Docker | **docker exec**

docker exec [OPTIONS] <CONTAINER> <COMMAND> [ARG...]
Executa um comando em um container em execução

Parâmetros	
-d	Executa o comando em backgroud
-e	Define variáveis de ambiente
-it	Entra em modo iterativo

Exemplo

```
$ docker exec -e MINHA_VARIAVEL=meuValor ubuntuLocal env | grep MINHA_VARIAVEL
> MINHA_VARIAVEL=meuValor

$ docker exec -d ubuntuLocal touch meuArquivo
$ docker exec AdvDBPostgres ls | grep meuArquivo
> meuArquivo

$ docker exec -it ubuntuLocal bash
```

Comandos Docker | **docker create**

docker create [OPTIONS] <IMAGE> <COMMAND> [ARG...]
Criar um novo container a partir de uma imagem

Parâmetros	
-e	Define uma variável de ambiente
--env-file	Local de uma arquivo com variáveis de ambiente
--link	Adicionar link a outro container
-m <quantidadeDeMemoria>	Define o limite de memória que o container pode usar do host
--memory-swap <quantidadeDeMemoria>	Define o limite de memória swap que o container pode usar (-1 para deixar ilimitado)
--name	Nome do container

Comandos Docker | **docker create**

Parâmetros	
-p	Mapeia uma porta entre o container e o host
--restart	Tipo de política de reinicialização do container
--rm	Remover o container automaticamente quando ele for terminado (default false) --runtime Tempo de execução para usar neste container
-v	Vincular um volume
--storage-opt	Opções do storage do container

Comandos Docker | **docker create**

Exemplo

```
$ docker create -e MINHA_VARIAVEL=minhaVariavel redis:latest
> 840bf6fa81cb026d15fd2c514e25bd5d2b36bea6f98650428adfd786eb559f3a

$ docker create -m 100m --memory-swap 120m redis:latest
> 01bf438ccf92043c3b67bfe06215b05ce6f898dd83d73465279ca3cbd7a97e61

$ docker create --rm -u root redis:latest
> 27b6da614858f24c9757b8a4f27d4d3526809b1592be725ba251b71d23960727

$ docker create -v $HOME/dev/temp:/docker --name --storage-opt size=10G meuRedis redis:latest
> 0471a8bd44ef5833ae81a5de8b6863451b5a1bc1047d5a34e7003720a4c34068

> docker create -p 8989:8989 redis:latest
> 7ab5a99ec88f1220ed4cf98fb5d7c265faed76ac6a3806219bd75fb74366417e
```

Comandos Docker | **docker stop**

docker stop [OPTIONS] <CONTAINER...>
Para um ou mais containers

Parâmetros	
-t <tempo>	Quantidade em segundos que o docker espera para parar o container

Exemplo

```
$ docker stop redisLocal
> redisLocal

$ docker stop -t 100 redisLocal
> redisLocal
```

Comandos Docker | **docker kill**

docker kill <CONTAINER...>
Mata um ou mais containers

Exemplo

```
$ docker kill redisLocal
> redisLocal
```

Comandos Docker | **docker ps**

docker ps [OPTIONS]
Lista os containers

Parâmetros	
-a	Mostrar todos os containers (por padrão mostra apenas os em execução)
-q	Exibi apenas os ids

Exemplo

```
$ docker ps
> CONTAINER ID    IMAGE    COMMAND    CREATED   STATUS          PORTS     NAMES
> 39e65a58f       redis    "docker"   16 sec    Up 15 seconds   6379/tcp  gallant

$ docker ps -a
> CONTAINER ID    IMAGE    COMMAND      CREATED       STATUS           PORTS  NAMES
> 39e65a58f       redis    "docker"     About a min   Exit 18 sec ago         gallant
> 9f40f0ffb       ubuntu   "/bin/bash"  12 min ago    Exit 12 min ago         mystifying

$ docker ps -q
> 39e65a58f
> 9f40f0ffb
```

Comandos Docker | **docker start**

docker start [OPTIONS] <CONTAINER...>
Inicia um ou mais containers

Exemplo

```
$ docker start redisLocal
> redisLocal
```

Comandos Docker | **docker restart**

docker restart [OPTIONS] <CONTAINER...>
Reinicia um ou mais containers

Parâmetros	
-t <tempo>	Quantidade em segundos que o docker espera para reiniciar o container, default são 10 segundos

Exemplo

```
$ docker restart redisLocal
> redisLocal

$ docker restart -t 100 redisLocal
> redisLocal
```

Comandos Docker | **docker stats**

docker stats

Exibir as estatísticas de uso containers

Exemplo

```
$ docker stats
> CONTAINER     CPU %   MEM USAGE / LIMIT       MEM %   NET I/O             BLOCK I/O     PIDS
> bbd788e30cce  0.08%   6.309 MiB / 7.704 GiB   0.08%   6.79 kB / 648 B     0 B / 0 B     3
> 237abe728103  0.07%   6.305 MiB / 7.704 GiB   0.08%   8.33 kB / 648 B     0 B / 0 B     3
> f4b319f2c636  0.06%   6.309 MiB / 7.704 GiB   0.08%   9.66 kB / 648 B     0 B / 0 B     3
```

Comandos Docker | **docker rename**

docker rename <CONTAINER> <NEW_NAME>
Renomeia um container

Exemplo

```
$ docker rename redisLocal redisCacheLocal
```

Comandos Docker | **docker rm**

docker rm [OPTIONS] <CONTAINER...>
Remove um ou mais containers

Parâmetros	
-f	Força a remoção, podendo remover containers em execução

Exemplo

```
$ docker rm 768e388fa2c9
> 768e388fa2c9

$ docker rm meuPhp
> meuPhp
```

Comandos Docker | **docker update**

docker update [OPTIONS] <CONTAINER...>
Atualiza as configurações um ou mais containers

Parâmetros	
-m <quantidadeDeMemoria>	Define o limite de memória que o container pode usar do host
--memory-swap <quantidadeDeMemoria>	Define o limite de memória swap que o container pode usar (-1 para deixar ilimitado)

Exemplo

```
$ docker update -m 1000m --memory-swap 1300m redisLocal
> redisLocal
```

Comandos Docker | **docker pause**

docker pause <CONTAINER...>
Pausar todos os processos dentro de um ou mais containers

Exemplo

```
$ docker pause redisLocal
> redisLocal
```

Comandos Docker | **docker unpause**

docker unpause <CONTAINER...>
Inicia os processos anteriormente pausados de um ou mais containers

Exemplo

```
$ docker unpause redisLocal
> redisLocal
```

Comandos Docker | **docker logs**

docker logs [OPTIONS] <CONTAINER>
Exibe os logs de um container

Parâmetros	
--tail <quantidadeLinhas>	Limita a quantidade de linhas que será exibido, começando do fim para o inicio
-f	Continua exibindo o log

Exemplo

```
$ docker logs --tail 5 redisLocal
> # Warning: no config file specified, using the default config. In order to specify a config file use
redis-server /path/to/redis.conf
> Redis 3.2.7 (00000000/0) 64 bit
> Running in standalone mode
> Port: 6379
> # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set
to the lower value of 128.
```

Comandos Docker | **docker rmi**

docker rmi [OPTIONS] <CONTAINER....>
Remover uma ou mais imagens

Parâmetros	
-f	Força remoção, podendo remover imagens com containers vinculados

Exemplo

```
$ docker rmi java
> Untagged: java:latest
> Untagged: java@sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
> Deleted: sha256:d23bdf5b1b1b1afce5f1d0fd33e7ed8afbc084b594b9ccf742a5b27080d8a4a8
> Deleted: sha256:0132aeca1bc9ac49d397635d34675915693a8727b103639ddee3cc5438e0f60
```

Comandos Docker | **docker commit**

docker commit [OPTIONS] <CONTAINER> [REPOSITORY[:TAG]]
Criar uma nova imagem a partir das alterações de um container existente

Parâmetros	
-a	Autor da commit
-m	Mensagem de commit

Exemplo

```
$ docker commit meuBancoDeDados
> sha256:37f6ba54d41c106df337d027a7345b582fe5b023aee1ca21c3719915d706b9e8

$ docker commit -a "andrejusti" -m "Container com registros já existentes" meuBancoDeDados
> sha256:393236a4cf00ab1295b07dbe09668b01c741888d502d98e5ccbb544a8abd043a
```

Comandos Docker | **docker tag**

docker tag <SOURCE_IMAGE[:TAG]> <TARGET_IMAGE[:TAG]>
Cria uma tag a partir de uma imagem existente

Exemplo

```
$ docker tag redis:latest redis:minhaTag
```

Comandos Docker | **docker push**

docker push [OPTIONS] <NAME>[:TAG]
Envia ao registro docker uma imagem

Exemplo

```
$ docker push minhaApp:releaseAws
> The push refers to a repository [docker-registro.meuservidor.com.br]
> 66d6e6240063: Layer already exists
> c35a5a30ce33: Layer already exists
> 656dc710f6e5: Layer already exists
> 8b8fbd29ed3e: Layer already exists
> fa4045f123ae: Layer already exists
> 8d705ae62407: Layer already exists
> c56b7dabbc7a: Layer already exists
> releaseAws: digest: sha256:b4ce72e1a87bf03d7cf870746e9a8c46c7b6e8c2 size: 2624
```

Comandos Docker | **docker build**

docker build [OPTIONS] <PATH | URL>
Criar uma imagem a partir de um Dockerfile, também é possível criar de um Dockerfile remoto, como em um repositório git no github

Parâmetros	
-f	Nome do arquivo de build (default é Dockerfile)
-t	Nome e opcionalmente uma tag no formato 'name: tag'

Comandos Docker | **docker build**

Exemplo

```
$ docker build localDockerFile
> Sending build context to Docker daemon 14.54 MB
> Step 1/4 : FROM anapsix/alpine-java --> 0e0d2021d670
> Step 2/4 : ADD target/app.jar /data/ --> c7ba906b9ba9
> Step 3/4 : EXPOSE 8080 --> c44bfacb345f
> Step 4/4 : CMD java -Djava.security.egd=file:/dev/./urandom $JAVA_OPTS -jar /data/app.jar --> a84461f1d4e5
> Successfully built a84461f1d4e5

$ docker build  -t minha-image:minha-tag -f Dockerfile.develop localDockerFile
> Sending build context to Docker daemon 14.54 MB
> Step 1/4 : FROM anapsix/alpine-java --> 0e0d2021d670
> Step 2/4 : ADD target/app.jar /data/ --> c7ba906b9ba9
> Step 3/4 : EXPOSE 8080 --> c44bfacb345f
> Step 4/4 : CMD java -Djava.security.egd=file:/dev/./urandom $JAVA_OPTS -jar /data/app.jar --> a84461f1d4e5
> Successfully built a84461f1d4e5
```

Dockerfile

Docker pode construir imagens automaticamente lendo as instruções do arquivo Dockerfile. O Dockerfile é um arquivo de texto que contém todos os comandos necessários para se criar uma imagem, usando o comando docker build podemos então criar a imagem a partir do Dockerfile.

Formato

```
# Comentario
INSTRUCAO argumentos
```

A instrução não faz distinção entre maiúsculas e minúsculas. Porém a convenção diz para as instruções serem MAIÚSCULAS para distinguir dos argumentos mais facilmente.

O Docker executa as instruções do Dockerfile em ordem. A primeira instrução deve ser “FROM” para especificar a Imagem Base da qual você está construindo.

Formato

```
# Meu teste
FROM ubuntu
```

Dockerfile | **FROM**

Informa a partir de qual imagem será gerada a nova image

Utilização

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

Exemplo

```
FROM ubuntu
FROM ubuntu:latest
FROM ubuntu@d2b1b8e4a217
```

Dockerfile | **MAINTAINER**

Campo opcional, que informa o nome do mantenedor da nova image, não recomendado, é recomendado usar o campo LABEL, onde é possível adicionar mais informações, o MAINTAINER no campo LABEL fica como LABEL maintainer "André Justi <justi.andre@gmail.com> <www.andrejusti.com.br>", assim quando é feito o comando docker inspect essa informação é mostrada

Utilização

```
MAINTAINER <name>
```

Exemplo

```
MAINTAINER "André Justi"
```

Dockerfile | **LABEL**

Adiciona metadados da imagem, informações adicionais que servirão para identificar versão, tipo de licença, mantenedor e etc

Utilização

```
LABEL <chave>=<valor> <chave>=<valor>
```

Exemplo

```
LABEL chave=valor

LABEL "chave"="valor"

LABEL "chave1"="valor1" "chave2"="valor2" "chave3"="valor3"

LABEL "chave1"="valor1" \
      "chave2"="valor2" \
      "chave3"="valor3"
```

Dockerfile | **ENV**

Instrução que cria e atribui uma variável de ambiente dentro da imagem. É possível informar mais de uma label

Utilização

```
ENV <chave>=<valor>
ENV <chave> <valor>
```

Exemplo

```
ENV chave=valor
ENV chave valor
```

Dockerfile | **WORKDIR**

Define qual será o diretório de trabalho (lugar onde serão copiados os arquivos, e criadas novas pastas etc)

Utilização

```
WORKDIR diretorio
```

Exemplo

```
WORKDIR /data
```

Dockerfile | **ADD** e **COPY**

Adiciona ou copia arquivos locais ou que estejam em uma url (no caso do ADD), para dentro da imagem

O <dest> é um caminho absoluto, ou um caminho relativo a WORKDIR, caso o diretório destino não exista, ele será criado
Se <src> for um diretório, todo o conteúdo do diretório será copiado, incluindo os metadados do sistema de arquivos

Utilização

```
ADD <src>... <dest>
ADD ["<src>",... "<dest>"] # Essa forma é necessário para caminhos que contêm espaços em branco

COPY <src>... <dest>
COPY ["<src>",... "<dest>"] # Essa forma é necessário para caminhos que contêm espaços em branco
```

Dockerfile | **ADD** e **COPY**

Exemplo

```
ADD minhaPasta/meuarquivo.txt /pastaRaiz/
ADD ["Área de trabalho/meu arquivo.txt", "pasta raiz"]


COPY minhaPasta/meuarquivo.txt /pastaRaiz/
COPY ["Área de trabalho/meu arquivo.txt", "pasta raiz"]
```

Dockerfile | **ADD** e **COPY**

Diferença entre ADD e COPY

- ADD Permite <src> ser um URL
- Se estiver executando um ADD é o <src> é um arquivo em um formato de compactação reconhecido, o docker irá descompactá-lo e irá copiar os arquivos para o <dist>

Exemplo

```
# Copiando arquivos compactados no formato tar para o container

# ADD
ADD resources/jdk-8.tar.gz /usr/local/

# COPY
COPY resources/jdk-8.tar.gz /tmp/
RUN tar -zxvf /tmp/jdk-8.tar.gz -C /usr/local
RUN rm /tmp/jdk-8.tar.gz
```

Dockerfile | **RUN**, **CMD** e **ENTRYPOINT**

Shell e Exec

As instruções RUN, CMD e ENTRYPOINT suportam duas formas diferentes de execução: o formulário shell e o formulário exec .

Exemplo Shell

```
INSTRUCAO executavel parametro01 parametro02
```

Exemplo Exec

```
INSTRUCAO ["executavel","parametro01","parametro02"]
```

Dockerfile | **RUN**, **CMD** e **ENTRYPOINT**

Ao usar o formulário shell, o comando especificado é executado com uma invocação do shell usando /bin/sh -c. É possível ver isso claramente ao executar um docker ps

Dockerfile exemplo

```
FROM ubuntu:trusty
CMD ping localhost
```

Execução

```
$ docker build -t nome:tag caminhoDockerfile

$ docker ps
> CONTAINER ID   IMAGE          COMMAND                  CREATED      STATUS      NAMES
> 52fec6512a86   teste:shell    "/bin/sh -c 'ping ..."   7 sec        Up 6 sec    gifted
```

Dockerfile | **RUN**, **CMD** e **ENTRYPOINT**

Uma opção melhor é usar o formulário exec nas instruções. Observe que o conteúdo do exec é formado como uma matriz JSON.

Quando a forma exec da instrução é usada, o comando será executado sem um shell.
OBS: É recomendado sempre usar o formulário exec nas instruções ENTRYPOINT e CMD.

Dockerfile exemplo

```
FROM ubuntu:trusty
CMD ["/bin/ping","localhost"]
```

Execução

```
$ docker build -t nome:tag caminhoDockerfile

$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED      STATUS      NAMES
0eab086cbe58   teste:exec     "/bin/ping localhost"    12 sec       Up 11 sec   gallant
```

Dockerfile | **RUN**, **CMD** e **ENTRYPOINT**

RUN: São as instruções que serão executadas para criação da imagem em questão, normalmente instalações de pacotes, esse comando também pode ser substituído ao iniciar o container, é possível declarar várias instruções RUN, porém cada uma cria uma nova camada da aplicação.

CMD: São instruções padrão do container, normalmente usadas para declarar como a aplicação deve iniciar, ou quais serviços deve iniciar ao executar, o CMD só pode ser declarado uma vez.

ENTRYPOINT : Específica o que será executado ao iniciar o container.

Dockerfile | **EXPOSE**

Expõem uma ou mais portas, isso quer dizer que o container quando iniciado poderá ser acessível através dessas porta

Utilização

```
EXPOSE <port> [<port>...]
```

Exemplo

```
EXPOSE 8080 8081
```

Dockerfile | **VOLUME**

Mapeia um diretório do host para ser acessível pelo container

Utilização

```
VOLUME ["/diretorio"]
VOLUME /diretorio
```

Exemplo

```
VOLUME ["/var/log"]
VOLUME /var/log
```

Dockerfile | **USER**

Define com qual usuário serão executadas as instruções durante a geração da image

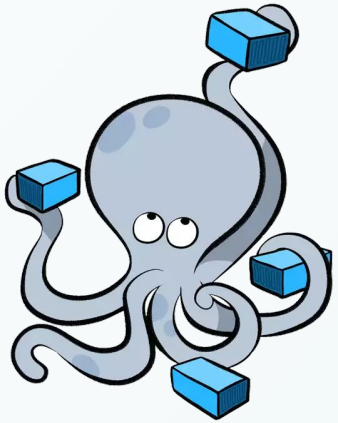
Utilização

```
USER usuario
```

Exemplo

```
USER root
```

Próximos passos | **Compose**

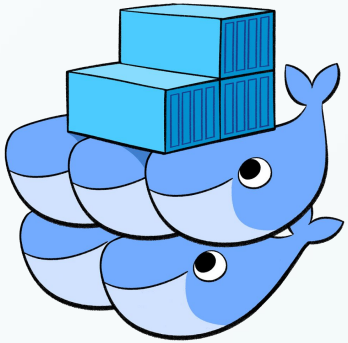


O Docker Compose é uma ferramenta para a criação e execução de múltiplos containers de maneira padronizada e que facilite a comunicação entre eles. Com o Compose, é possível usar um único arquivo para definir como será o ambiente de uma aplicação e usando um único comando possível criará e iniciará todos os serviços definidos.

Próximos passos | **Machine**

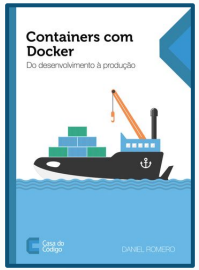
Docker machine é a ferramenta usada para instalação e gerência de docker hosts remotos de forma fácil e direta.

Próximos passos | **Swarm**

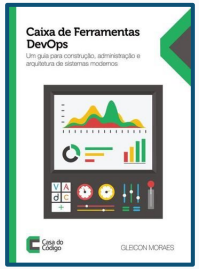


O Docker Swarm é uma ferramenta que permite a criação de clusters de Docker, ou seja, podemos fazer com que diversos hosts de Docker estejam dentro do mesmo pool de recursos, facilitando assim o deploy de containers. É possível por exemplo criar um container sem necessariamente saber em qual host ele está, Swarm disponibilidade uma API de integração, onde é possível realizar grande parte das atividades administrativas de um container

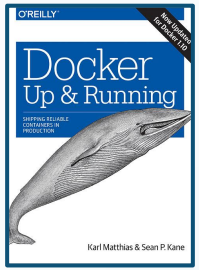
Referências



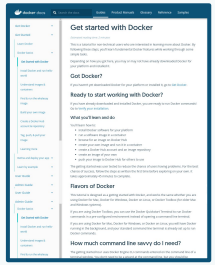
Livro: **Containers com Docker**
Do desenvolvimento à produção
Casa do Código



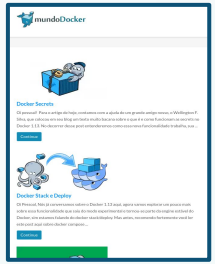
Livro: **Caixa de Ferramentas DevOps**
Um guia para construção, administração e arquitetura de sistemas modernos
Casa do Código



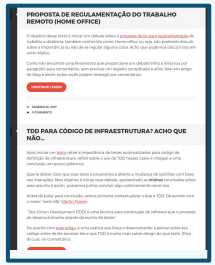
Livro: **Docker: Up & Running**
Shipping Reliable Containers in Production
O'Reilly



Site: **https://docs.docker.com**
Documentação oficial Docker



Site: **http://mundodocker.com.br**
Site de uma comunidade brasileira



Site: **http://techfree.com.br**
Site com várias dicas e posts sobre Docker

OBRIGADO!!!

André Justi
justi.andre@gmail.com
+55 48 996542190

docker
DEV.SAJ ADV