Fundamentos do Docker

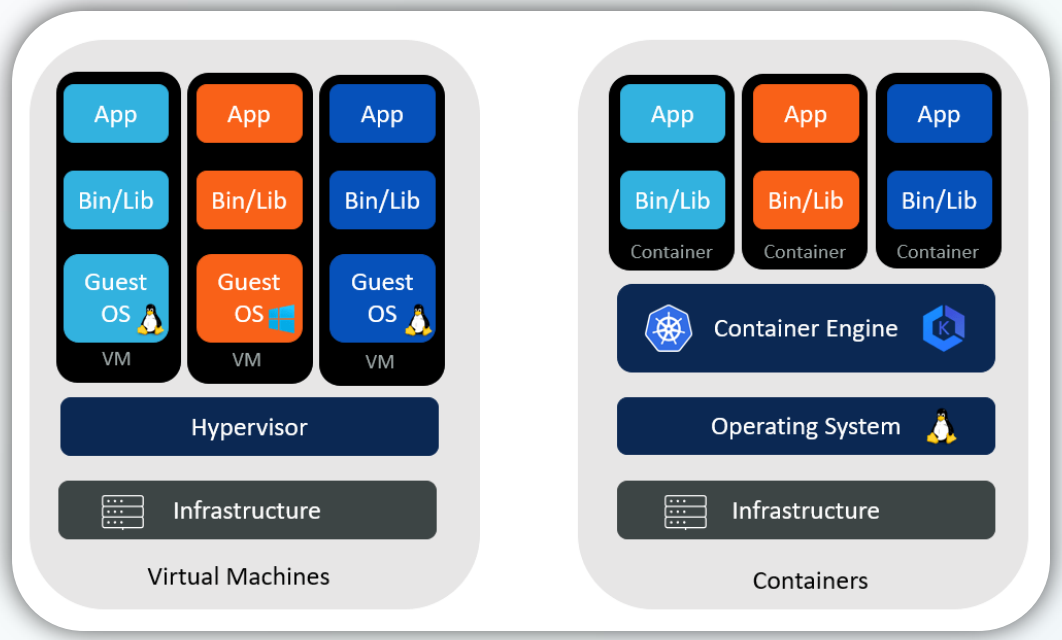
Patrick Alves
https://cognicode.tech

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 **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 Docker foi criado em 2013 por Solomon Hykes como uma plataforma open-source para desenvolvedores.

Virtualização Tradicional VS Containers

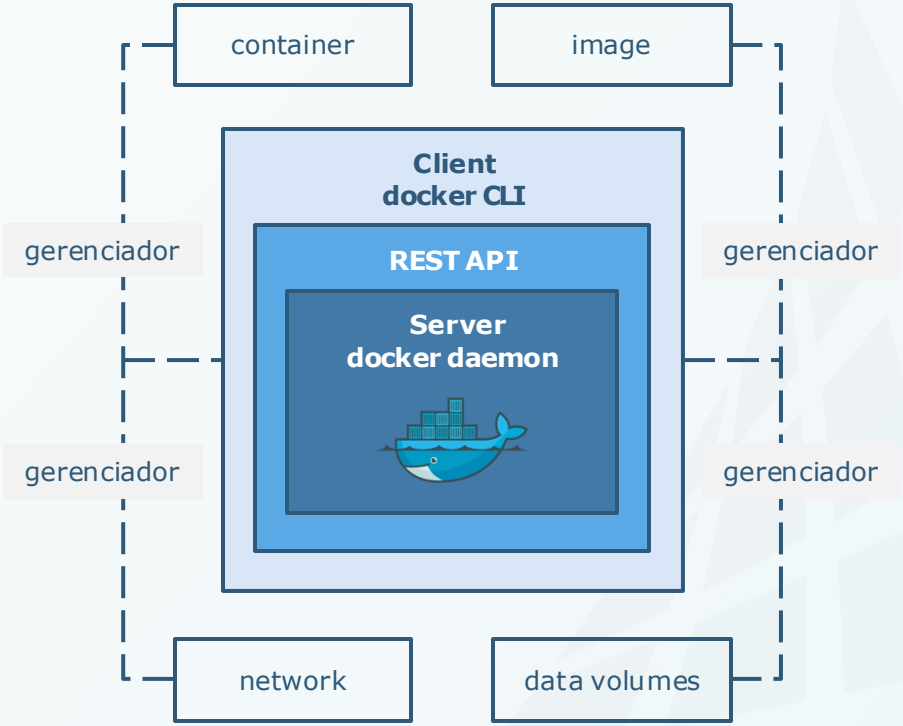
- Virtualização tradicional: envolve a criação de uma máquina virtual que emula um sistema operacional completo
- Containers: usa o sistema operacional host e compartilha recursos com outros containers



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.

- ✓ **Daemon:** um processo que roda no sistema operacional host.
- ✓ **Client**: interface de linha de comando ou GUI para gerenciar o Docker.
- ✓ **Imagens**: que são pacotes que incluem o código, dependências e outras configurações necessárias para executar um aplicativo.

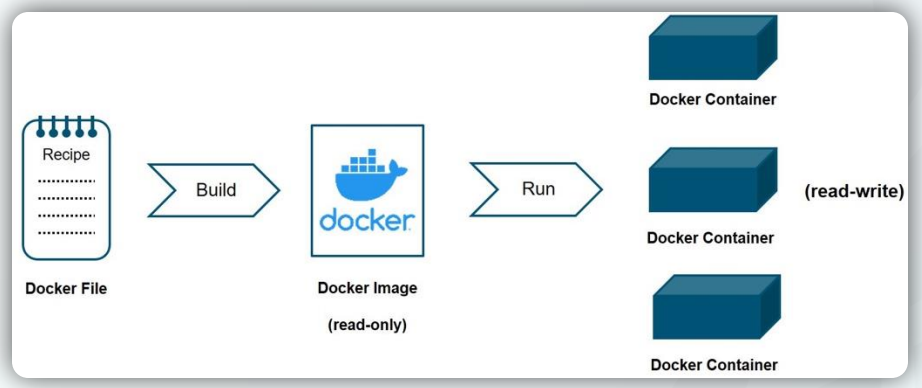


Elementos do **Docker**

Containers docker - Containers tem como base sempre uma **imagem**, pense como na seguinte analogia do mundo Python, uma imagem é uma classe e um container é como um objeto instância dessa classe, então podemos através de uma imagem “instanciar” vários containers.

Algumas características dos containers

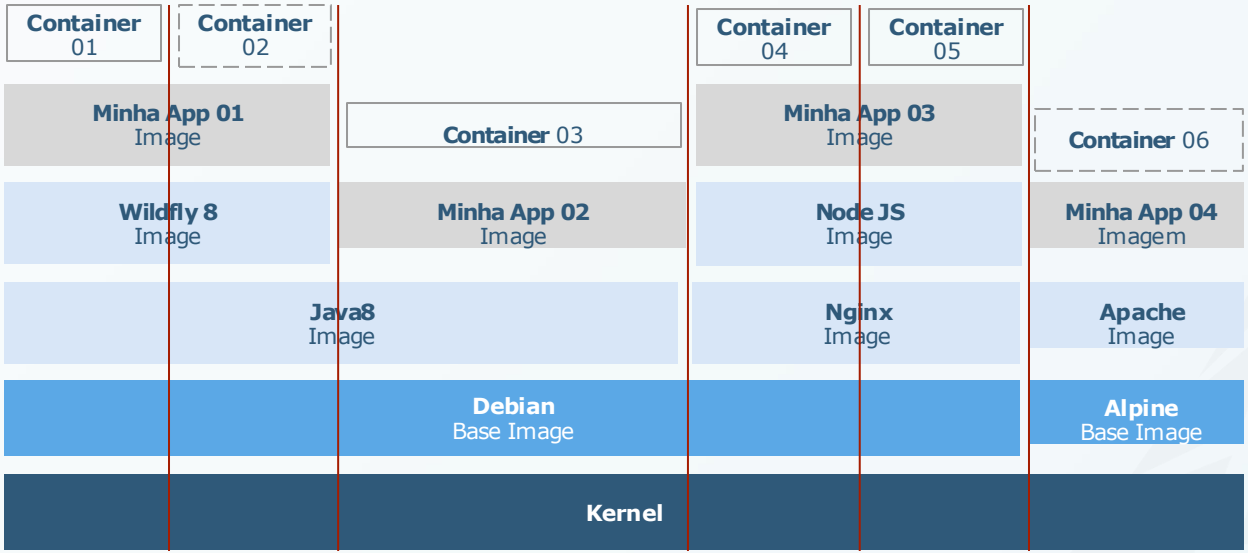
- Portabilidade de aplicação
- Isolamento de processos
- Prevenção de violação externa
- Gerenciamento de consumo de recursos.



Elementos do **Docker**

imagens docker - Imagens são templates para criação de containers.

- Imagens são imutáveis, para executá-las é necessário criar uma instância: (container),
- Imagens são construídas em camadas, o que facilita sua reutilização e manutenção.



Elementos do **Docker**

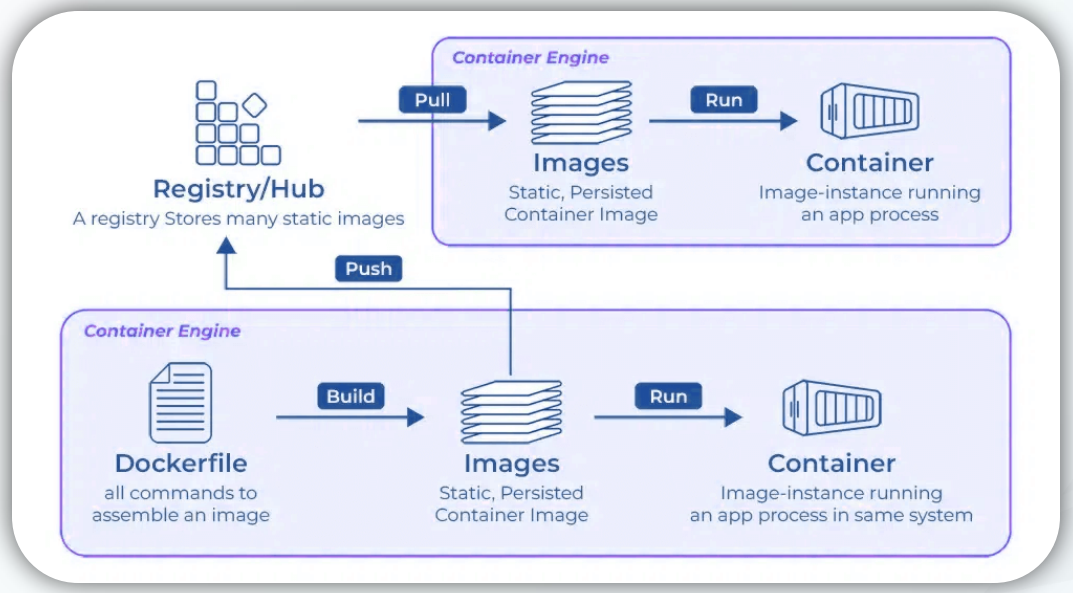
Dockerfile - São scripts com uma série de comandos para criação de uma imagem, nesses scripts podemos fazer uma séries de coisas como executar comandos sh, criar variáveis de ambiente, copiar arquivos e pastas do host para dentro da imagem etc.

Exemplo

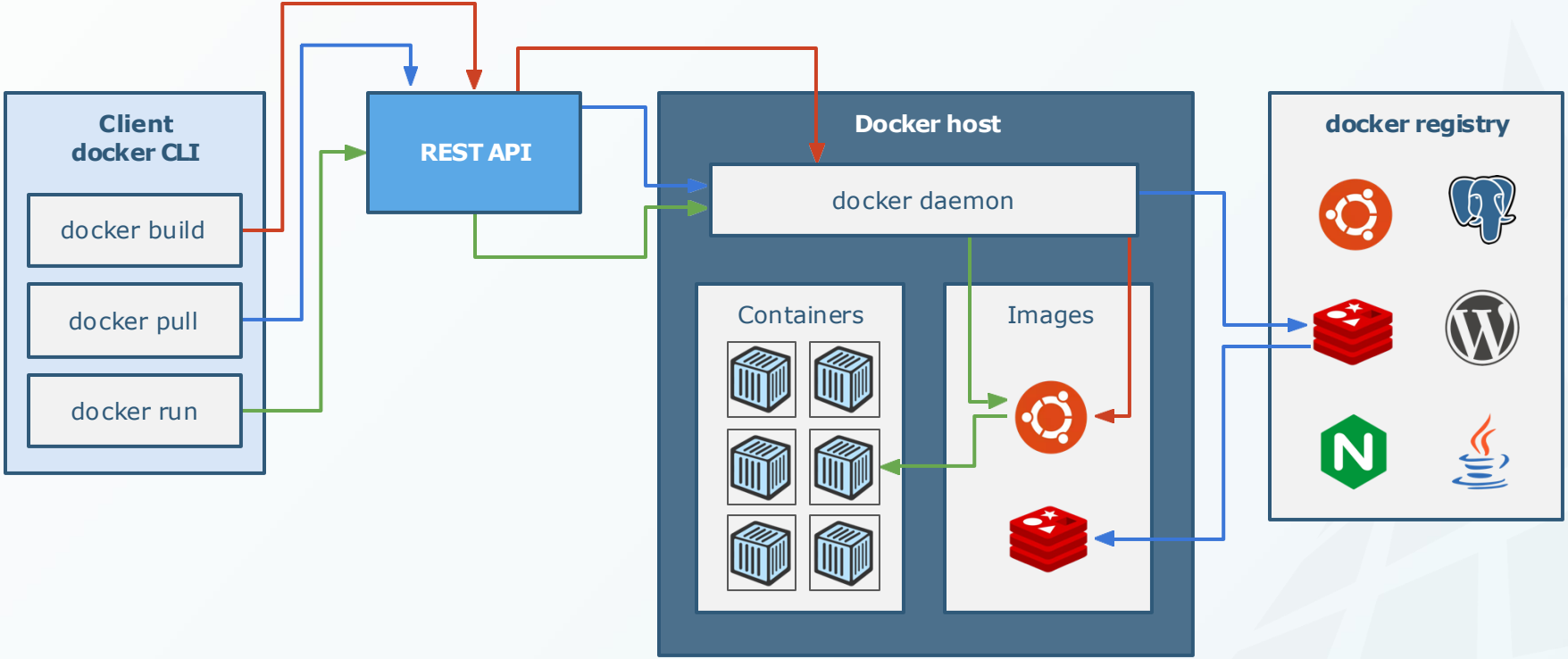
```
FROM ubuntu

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 | **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
```

teste

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.
- Quando você configura um link, você cria um elo de ligação entre um container de origem e um
- 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**

Docker Volumes

- É a única maneira de preservar os dados do container em execução
- Através do volume é possível mapear uma pasta no host para uma pasta interna do container.

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 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 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
```

Dockerfile

O Dockerfile é um arquivo de texto que contém todos os comandos necessários para se criar uma imagem.

```
# Comentario
INSTRUCAO argumentos
```

- ✓ A instrução não faz distinção entre maiúsculas e minúsculas (use MAIÚSCULAS)

- ✓ 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.

```
# Meu teste
FROM ubuntu
```

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 | **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 | **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
```

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'

Exemplo API Python

https://github.com/cpatrickalves/curso-docker/tree/main/codigos/api01

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

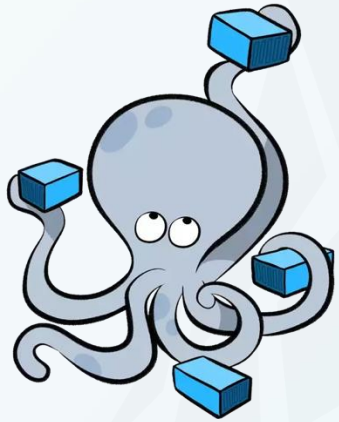
Exemplo

```
$ 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
```

Docker **Compose**

O Docker Compose é uma ferramenta para a criação e execução de ***múltiplos containers de maneira padronizada*** *e que facilita a* ***comunicação*** *entre eles.*

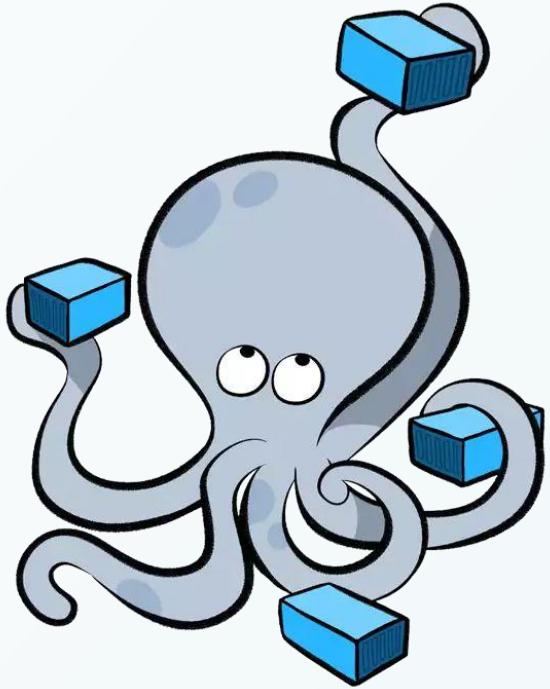
- Usa um arquivo **YAML** (`docker-compose.yml`) para configurar serviços, redes e volumes.
- Simplifica a execução e orquestração de múltiplos containers com um único comando.



Quando usar?

- Quando sua aplicação consiste em vários containers (ex: frontend, backend, banco de dados).
- Para configurar dependências entre containers (como garantir que o banco de dados inicie antes da aplicação).
- Para facilitar a colaboração entre equipes com um ambiente padronizado.

Docker **Compose**



Arquivo `docker-compose.yml`

Os arquivos YAML são documentos de **texto** que organizam dados em uma **estrutura hierárquica** usando pares de **chave-valor, listas e mapeamentos**. A sintaxe é baseada em **indentação**, onde espaços em branco indicam a hierarquia dos dados.

Útil para executar um conjunto de containers **em um único host.**

```yaml
---
name: John Doe
age: 30
address:
  street: 123 Main St
  city: Springfield
  state: IL
  zip: 12345
phone_numbers:
  - 123-456-7890
  - 987-654-3210
...
```

Arquivo `docker-compose.yml`

Estrutura básica:

- **Versão:** Define a versão do arquivo Compose.
- **Services:** Lista de containers que compõem a aplicação.
- **Networks e Volumes:** Definição de redes e volumes a serem compartilhados.

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

Arquivo `docker-compose.yml`

Comandos:

- **docker-compose up** - Inicia todos os serviços definidos no arquivo.
- **docker-compose down** - Para e remove containers, redes e volumes.
- **docker-compose ps** - Exibe o status dos containers definidos em um arquivo docker-compose.yml
- **docker-compose logs** - Exibe logs dos containers.
- **docker-compose exec** - Executa comandos dentro de um container.

Arquivo docker-compose.yml

```
docker run mmumshad/simple-webapp
docker run mongodb
docker run redis:alpine
docker run ansible
```

docker-compose.yml

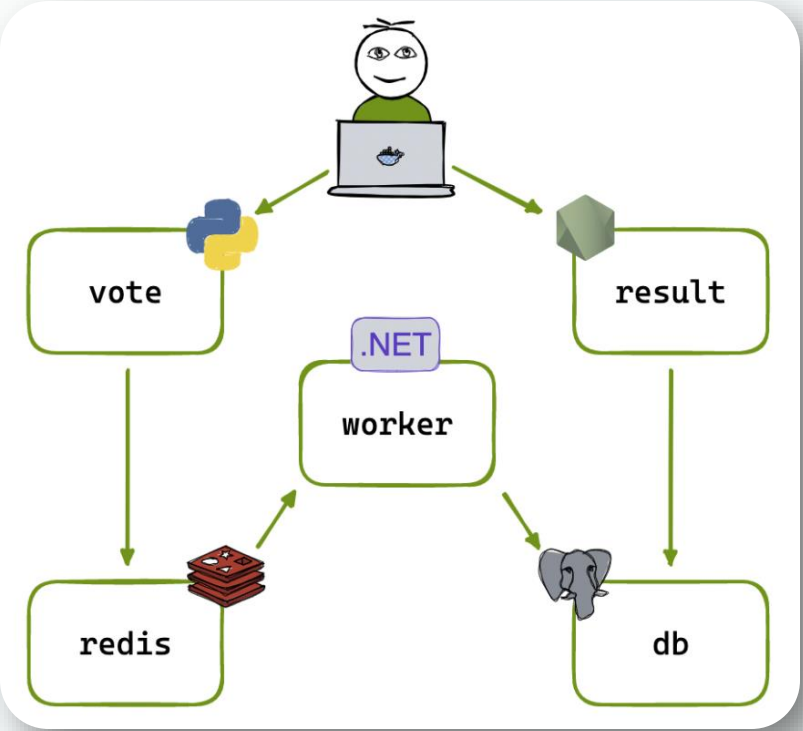
```
services:
    web:
        image: "mmumshad/simple-webapp"
    database:
        image: "mongodb"
    messaging:
        image: "redis:alpine"
    orchestration:
        image:  "ansible"
```

Exemplo API Python com Docker Compose

https://github.com/cpatrickalves/curso-docker/tree/main/codigos/api01

Exemplo Voting App

https://github.com/cpatrickalves/curso-docker/tree/main/codigos/voting-app

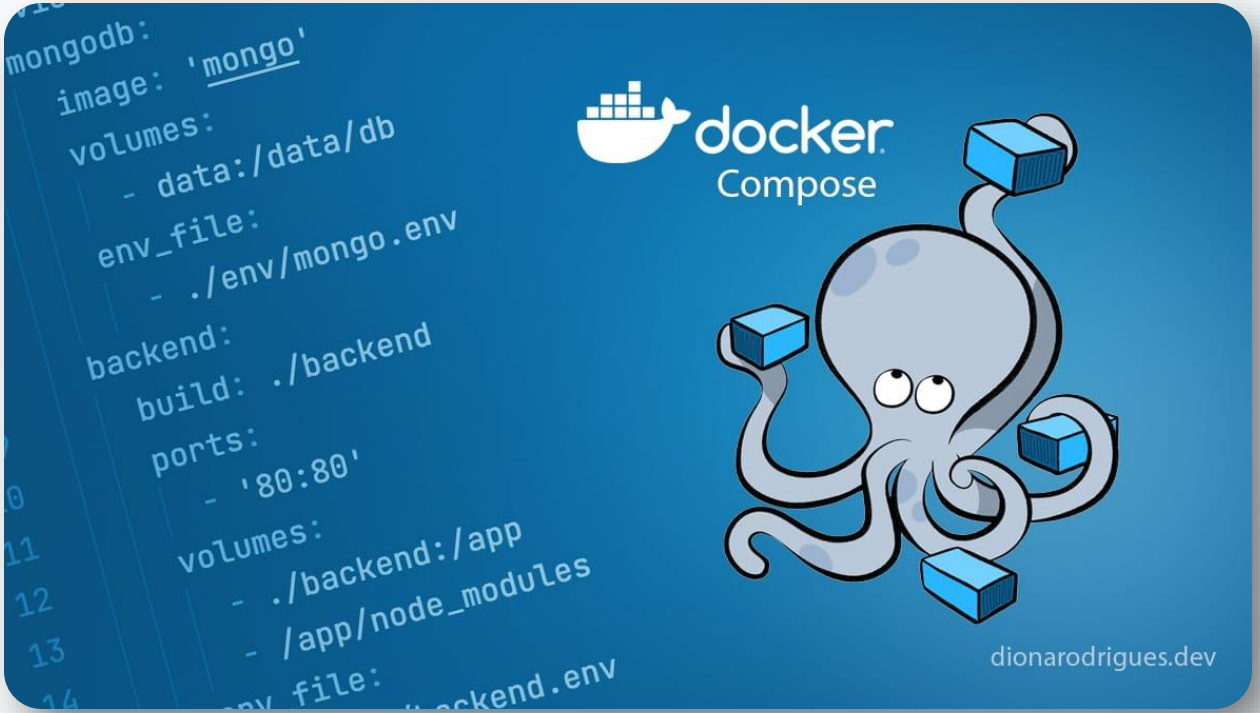


Arquitetura:

- **WebApp** – feito em Python que permite votar**.**
- **Redis** – coleta os votos.
- **Worker** – Consome os votos e armazena no banco.
- **Db** – PostgreSQL com um volume.
- **Result –** aplicação em Node.js para mostrar os resultados.

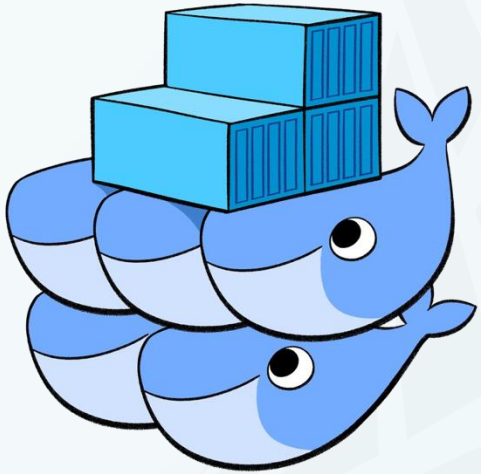
Exemplo Voting App – **Docker Compose**

https://github.com/cpatrickalves/curso-docker/tree/main/codigos/voting-app



Docker **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 recurso**s, facilitando assim o deploy de containers.



Vantagens

- Aumenta a disponibilidade (no single point-of-failure)
- Aumenta escalabilidadea
- **Worker –** Consome os votos e armazena no banco.
- **Db** – PostgreSQL com um volume.
- **Result –** aplicação em Node.js para mostrar os resultados.

Docker **Swarm**

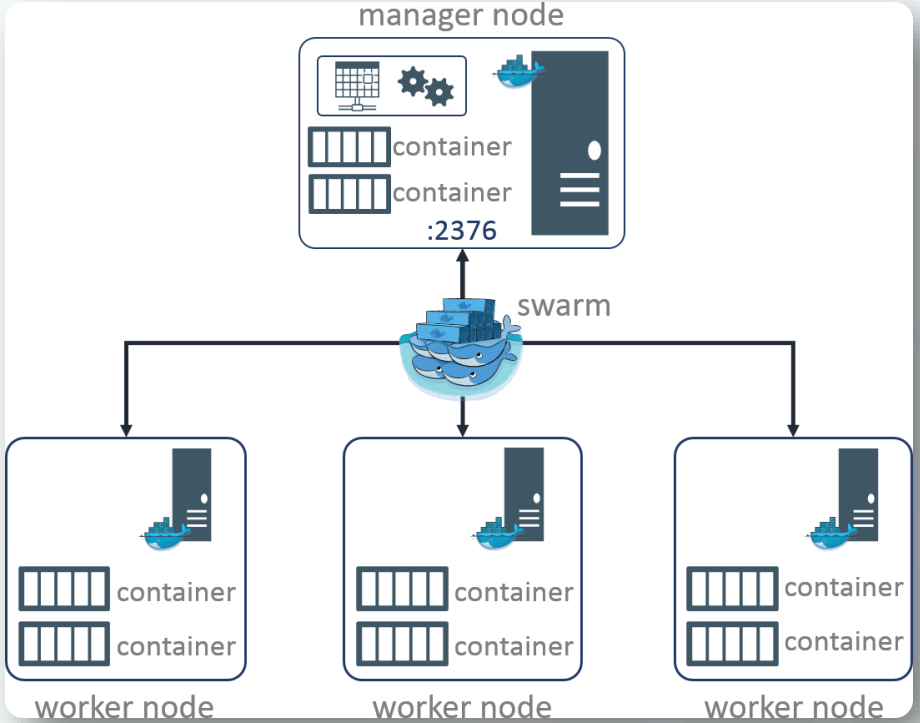
Vantagens

- Aumenta a disponibilidade (*no single point-of-failure*)
- Aumenta escalabilidade.
- Permite balanceamento de carga

Nodes

- **Manager**: Responsável por gerenciar o estado do cluster, coordenar a execução dos serviços e garantir que a configuração desejada seja mantida.
- Os nós manager também podem executar contêineres.

- **Worker**: Executam os contêineres conforme as instruções dos nós manager.
- Eles não gerenciam o estado do cluster, mas reportam seu status aos managers.



Criando um **Swarm**

Inicializa o Swarm

```
$ docker swarm init --advertise-addr <server_ip_address>
```

Mostra o comando de Join no Node master

```
$ docker swarm join-token worker
```

Adiciona node no Cluster Swarm

```
$ docker swarm join --token <token> <MasterIP>:<MasterPort>
```

Remove o Node do Swarm (no worker)

```
$ docker swarm leave
```

Remove o Node do Swarm (no Master)

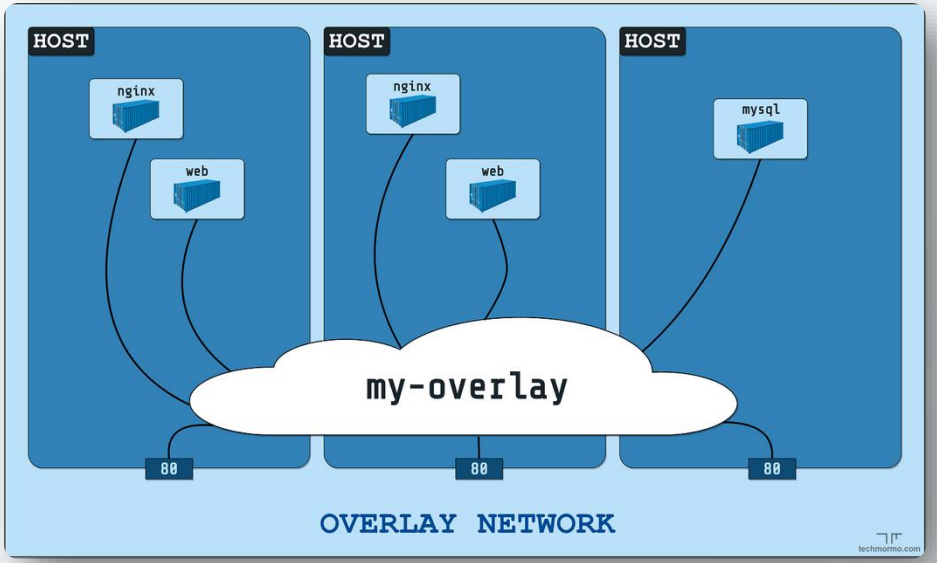
```
$ docker node rm <node-name>
```

Overlay Network

A overlay network é uma rede virtual distribuída que permite a comunicação entre contêineres em diferentes nós (máquinas) dentro do cluster Swarm.

Funcionalidades:

- Conectividade entre nós
- Isolamento de Rede
- Comunicação entre Serviços



Criando rede Overlay

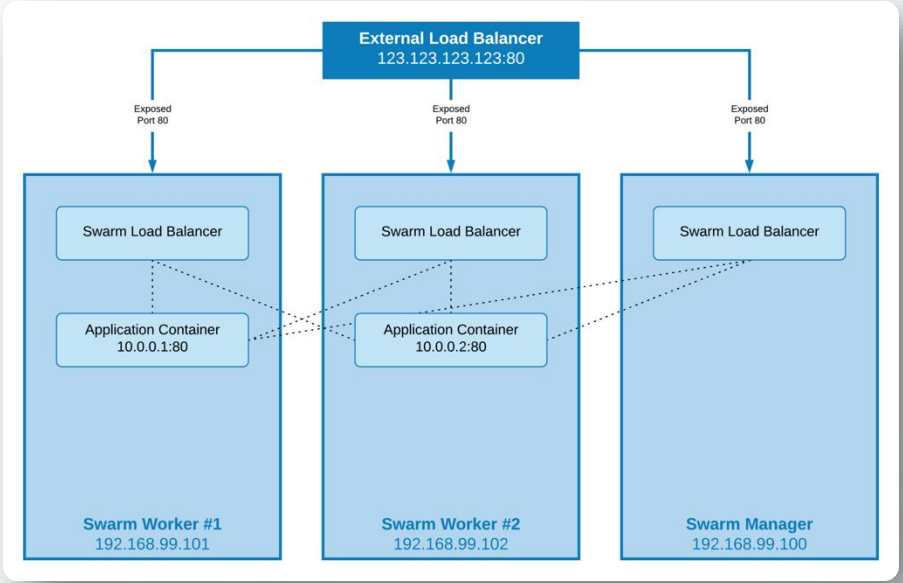
```
$ docker network create -d overlay nome_da_rede
```

Ingress Network

A ingress network é uma overlay network especial, criada automaticamente pelo Docker Swarm para gerenciar o load balancing (balanceamento de carga) de serviços que estão expostos externamente.

Funcionalidades:

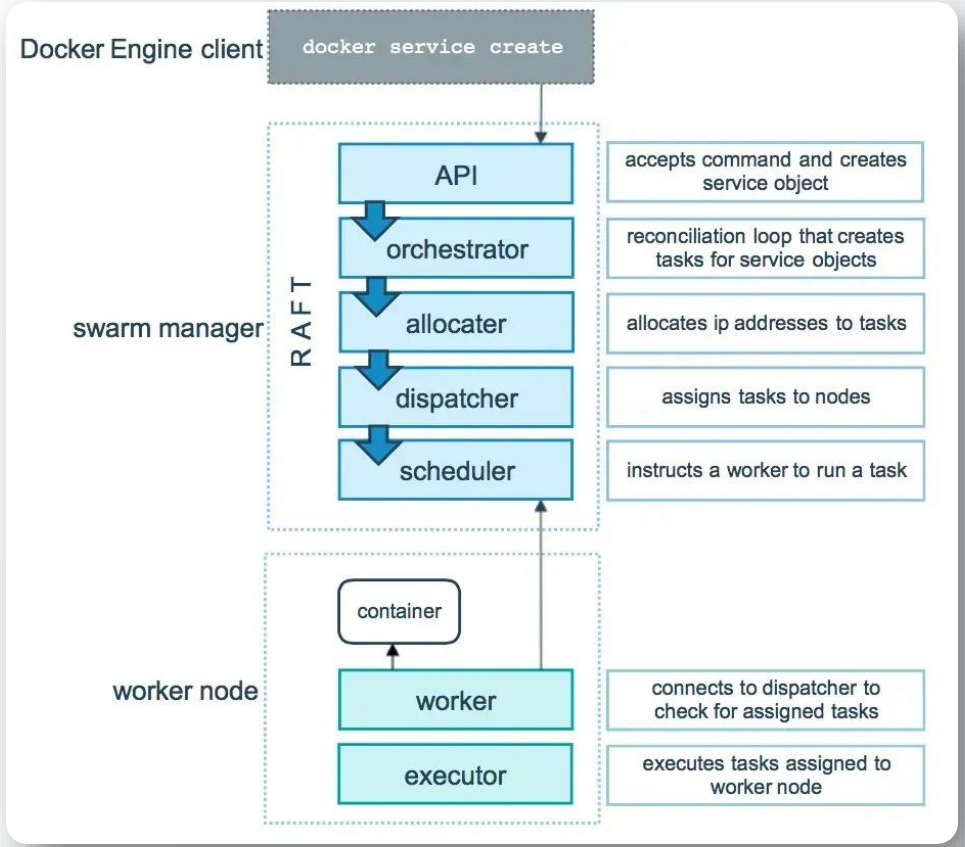
- Balanceamento de Carga
- Roteamento de Entrada
- Criada Automaticamente



Exemplo:

```
$ docker service create --name meu-servico --publish 80:80 nginx
```

Arquitetura do Docker **Swarm**



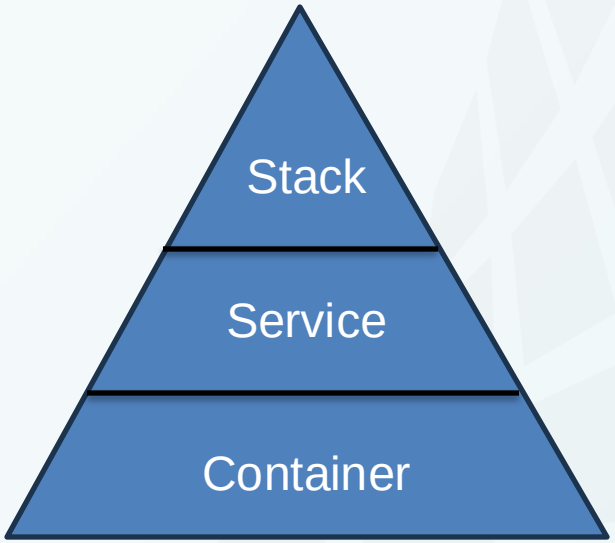
Docker **Swarm**

Services

- Representam as **aplicações** que serão executadas no cluster. Um serviço pode ser composto por múltiplas instâncias de um contêiner, permitindo escalabilidade.

Stacks

- Agrupamentos de serviços que se comunicam entre si, permitindo a gestão de aplicações complexas compostas por múltiplos serviços.
- Permite a criação de Réplicas
- Permite a configuração de Limites de CPU, Memória, acesso a GPU etc.
- Evolução do Docker Compose (version 3), toda configuração em um único arquivo.
- Arquivo YAML.



Docker **Swarm - Stacks**

```yaml
services:
  redis:
    image: redis:alpine
    networks:
      - frontend
    deploy:
      replicas: 1
      resources:
        limits:
          cpus: "0.1"
          memory: 50M

  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      replicas: 2
      placement:
        constraints:
          - node.role == manager
```

Docker Swarm - Aplicações para Gestão do Cluster

Traefik

- O Traefik é um **proxy reverso** e balanceador de carga projetado para facilitar a gestão de tráfego em aplicações web, especialmente em ambientes dinâmicos como o Docker.
- Se destaca pela sua capacidade de detectar automaticamente serviços e containers, configurando o roteamento de tráfego sem a necessidade de intervenção manual constante.

Portainer

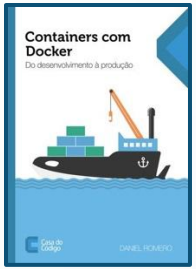
- Portainer é uma aplicação web de código livre projetada para facilitar o gerenciamento de contêineres Docker e outras plataformas de orquestração, como Swarm e Kubernetes.
- Transforma a administração de contêineres, que tradicionalmente é realizada por linha de comando, em um processo mais intuitivo e visual.

Docker **Swarm - Resumo**

Etapas

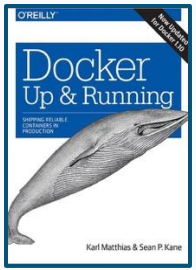
- Instalação do Docker
- Inicialização do Cluster Swarm
- Criação da Overlay network
- Adicionar Workers
- Configurar DNS
- Adicionar o Traefik
- Adicionar o Portainer
- Instalar as aplicações via Portainer

Referências

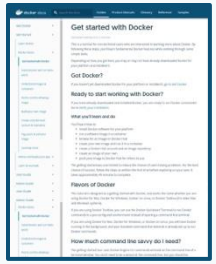


Livro: **Containers com Docker**
Do desenvolvimento à produção **Casa do Código**

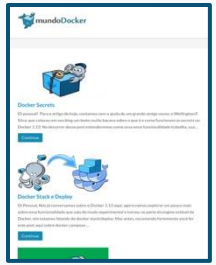
Livro: **Caixa de Ferramentas DevOps**
Um guia para construção, administração e arquitetura de sistemas modernos
Casa do Código



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

teste

OBRIGADO!!!

Patrick Alves
patrick@cognicode.tech

https://cognicode.tech

teste