Introdução
Docker

Docker

O Docker é um programa que realiza **virtualização no nível do sistema operacional**, também conhecido como contêiner, especificamente para **aplicações** do sistema.

É desenvolvido pela Docker, Inc.

O Docker foi desenvolvido principalmente para Linux, onde utiliza os principais recursos de isolamento do kernel Linux (cgroups / namespaces), systemd. Permitindo que "contêineres" independentes sejam executados dentro de um uma única instância do Linux, dando a visão de como se tivesse diversos sistemas isolados.

https://docs.docker.com

Introdução
Docker

Docker

O suporte do kernel Linux para namespaces principalmente isola a visão de um aplicativo (processo) do ambiente do sistema operacional.

- Incluindo árvores de processo
- Rede
- IDs de usuário
- Sistemas de arquivos montados
- CGroups do kernel fornecem recursos limitadores de memória e CPU.

Desde a versão 0.9, o Docker inclui a biblioteca **libcontainer** como uma maneira de usar diretamente os recursos de virtualização fornecidos pelo kernel do Linux, além de usar interfaces de virtualização abstratas via libvirt, LXC, systemd-nspawn e containerd.io.

Introdução
Docker

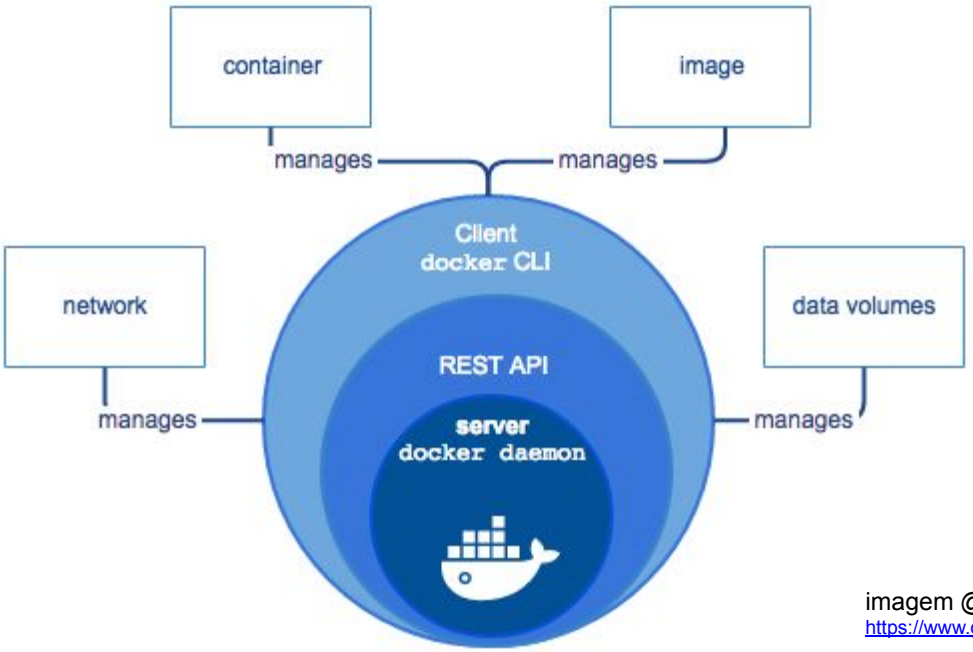


imagem @Google
https://www.georgestudenko.com/

Introdução
Docker

Docker

- Utiliza funcionalidades de isolamento de recursos do Kernel de Linux
- Cada container executa um conjunto de processos separado
- Consome recursos nativos do hardware sem a camada de virtualização e do SO.
- O processo é “virtualizado”, mas o processamento não é virtualizado.

O suporte no Windows atualmente pode ser nativo:

- Windows usa o Hyper-V e Containers Windows features (bin Windows ou Linux).

O suporte no MAC atualmente não é nativo:

- MAC-OS xhyve

(Virtualiza o Docker Engine e as características do kernel de Linux para o Docker daemon)

Introdução
Docker

Introdução Docker

- Ambientes de desenvolvimento comuns (Desenvolvimento / Infraestrutura)
- Deploy das aplicações com maior facilidade e de forma automatizada.
- Agilidade, controle e portabilidade na gestão dos ambientes.
- (Ex. MySQL, PHP, Solr, etc)

Cada serviço é representado por um container ou até mesmo em um único container específico.

Introdução
Docker

Conceitos de Docker

Docker, Docker API, Images, Containers, DockerHub e Dockerfile.

Compose / Stack e Swarm.

Software para Contêiner : LXC / LXD e Docker
Docker

Docker

```
# apt install docker.io # (docker empacotado pela distribuição)
```

Docker CE (ultima versão)

```
# curl https://www.ic.unicamp.br/~william/howto/install/ubuntu/docker > docker.sh
# bash -x docker.sh
# docker -v
# systemctl status docker

# docker -v
Docker version 19.03.12, build 48a66213fe

# docker info
```

Introdução API
Docker

Instalação:

Docker Engine (API)

```
# netstat -lnp | grep docker.sock
unix  2      [ ACC ]     STREAM     LISTENING     148262   1/systemd
  /var/run/docker.sock


$# docker cli utiliza a API da engine Docker através de um socket UNIX.
$# docker -> /var/run/docker.sock
```

Mudando API para um socket TCP

https://greencircle.vmturbo.com/docs/DOC-4710-how-to-specify-ports-that-the-docker-api-listens-on

Configurando o Docker Cli

https://docs.docker.com/engine/reference/commandline/cli/

```
export DOCKER_HOST=143.106.16.156:2323
```

Introdução
Docker

Imagem Docker

Imagens Docker prove as aplicações
(binários, bibliotecas, execuções e pré-configuração dos ambientes)
podem ser criadas por qualquer pessoas, ou disponibilizada pela equipe de desenvolvimento oficial.

* Imagens podem ser criadas somente com suas aplicações.
* Ou existir uma imagem base -> seu conteúdo -> sua imagem.

Introdução repositório de imagem oficial
Docker

Docker Hub (https://hub.docker.com/)

Dev-test pipeline automation, 100,000+ free apps, public and private registries

O lugar ideal para encontrar imagens atualizadas e com suporte oficial ou até mesmo imagens criada pela comunidade é o DockerHub, devido à popularidade do Docker.

Esta plataforma tem repositórios oficiais de Apache, MongoDB, Nginx, WordPress entre outros, prontos para serem utilizados.

Além disso, nesta plataforma, você poderá criar repositórios para suas imagens, se assemelhando ao GitHub ou GitLab, onde você pode criar múltiplos repositórios públicos, mas apenas um repositório privado. No caso de você precisar de mais, existe a possibilidade de pagar uma assinatura.

Repositório privado de imagens
Docker

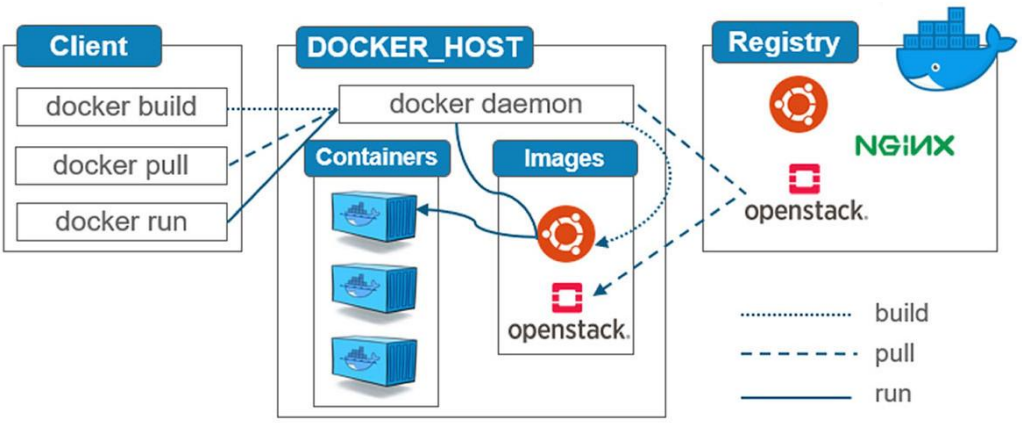


imagem @Google

Docker Private Registry

O Registry é um aplicativo altamente escalável que permite armazenar e distribuir imagens Docker.
https://docs.docker.com/registry/

Como instalar

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04

Introdução Imagens
Docker

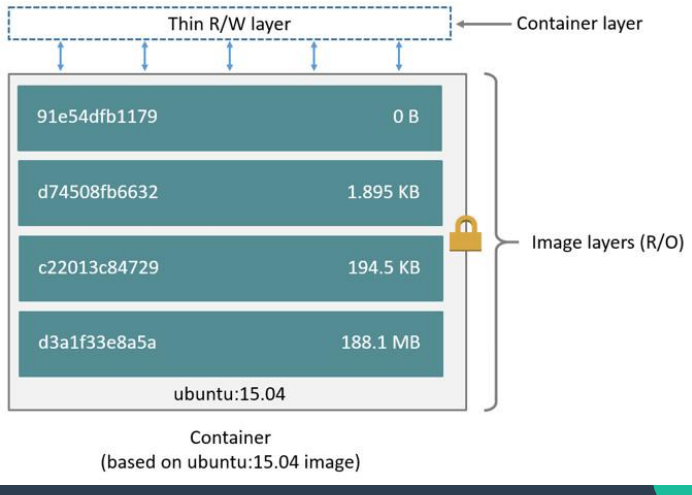
Imagem Docker e Container

Uma imagem refere-se a uma lista de camadas, que são empilhadas uma acima da outra.

Note que a imagem é imutável, mas facilmente estendida.

O container, por outro lado, é uma instância no tempo de execução de uma imagem.
Quando um novo container é criado, uma nova camada de escrita é criada no topo das camadas adjacentes.
Todas as alterações feitas no container em execução são feitas na mesma.

Uma vantagem dos containers Docker é a portabilidade, pois nos permite utilizar a mesma imagem em diferentes distribuições Linux com configurações distintas de hardware sem alterar as imagens.



Imagens
Docker

Exemplo Dockerfile: (meu sistema)

```
FROM httpd
COPY index.html /var/www/html/
CMD httpd-foreground
```

Compilando minha imagem com index.html de uma imagem base.

```
# docker build -t willreli/http-com-meu-index:latest
```

Camadas:
* httpd (tem várias camada tbm)
* copy index.html (segunda camada)
* cmd (terceira camada)

```
# docker history willreli/http-com-meu-index:latest
```

Gerenciando imagens do repositório oficial.
Docker

https://hub.docker.com/

Administrando imagens

- Fazer download de uma imagem (oficial/comunidade).

```
# docker pull <nome-imagem-dockerhub>
```

* Pode-se utilizar dois pontos para definir uma versão específica da imagem.

```
# docker pull <nome-imagem-dockerhub>:<versão>
```

- Lista de imagens no local (baixados e criados).

```
# docker images
```

- Apagar imagens

```
# docker rmi <nome-imagem-dockerhub>
```

helpppppppp
Docker

```
# docker help image (todos os comandos docker)

Usage:   docker image COMMAND

Manage images

Options:
      --help   Print usage

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

docker run ...
Docker

Criar e iniciar um container docker baseada na imagem que está localmente, executar um comando:

```
# docker run -it --name <nome-container> <nome-imagem> <comando>
```

Exemplos:

```
# docker run -it --name my-container node bash
```

Criar e iniciar um container, executar comando e destruir o container

```
# docker run --rm -it --name my-container node bash
```

* É recomendável utilizar o parâmetro --rm , desta forma o container é removido automaticamente quando terminar sua execução.

```
# docker run -it -d --name teste6 centos bash
```

docker run ...
Docker

Executa com terminal interativo e detach (-t terminal / -i interativo / -d “*desatachado”)*

```
# docker run -it -d --name teste6 centos bash
```

Docker exec (novo processo) ou senão dependendo da aplicação você consegue atachar novamente (programa interativo ou debug)

```
# docker attach --detach-keys 'ctrl-a' teste7
```

help run ...
Docker

Docker run parâmetros:

```
# docker help run

...
-i, --interactive  Keep STDIN open even if not attached
...
-t, --tty              Allocate a pseudo-TTY
...
-p, --publish list Publish a container's port(s) to the host (default [])
...
-P, --publish-all  Publish all exposed ports to random ports
```

Processos
Docker

```
# docker ps -a (lista container ativo e finalizados)

# docker top <nome-ou-id-container>

# docker exec -it 3a22d3c67adc bash
 - > sleep 9000

  ├─dockerd -H fd://
  |   ├─11*[{dockerd}]
  |   └─containerd -l
unix:///var/run/docker/libcontainerd/docker-containerd.sock
--metrics-interval=0 --start-timeout 2m ...
  |       ├─10*[{containerd}]
  |       ├─containerd-shim
3a22d3c67adc3b0f7a2547d2c96cd018a08aad4d7d679c0380dd74e7dfd49338...
  |       |   ├─7*[{containerd-shim}]
  |       |   └─bash
  |       └─containerd-shim
3a22d3c67adc3b0f7a2547d2c96cd018a08aad4d7d679c0380dd74e7dfd49338...
  |           ├─7*[{containerd-shim}]
  |           └─bash
  |               └─sleep 9000
```

Manipulando o seu container
Docker

Matar todos os containers em execução

```
# docker kill $(docker ps -q)
```

Parar container

```
# docker stop <id-ou-nome-container>
```

Eliminar todos os containers suspensos

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

Obter o IP de um container

```
# docker inspect container_name
```

Iniciando novamente o container no estado atual.

```
# docker start 3a22d3c67adc
# docker start -ai 3a22d3c67adc
```

Alguns macetes ;)
Docker

Onde fica os arquivo do seu container

- overlay2

```
# docker inspect <ID/NOME CONTAINER>
```

/var/lib/docker/overlay2/XXXXXXXXXXX
./merged -> arquivos da imagem default (habilitado quando ativo)
./diff -> "seus arquivos e mudanças"

- Vinculando Volume (compartilhando) :

```
# docker run -dit --name my-apache-app -p 8080:80 \
-v /tmp/documentroot:/usr/local/apache2/htdocs/ httpd
```

Extra Docker

Visualizar a diferença de arquivos entre a imagem base e container

docker diff <container>

A : A file or directory was added
D : A file or directory was deleted
C : A file or directory was changed

Alguns macetes ;)
Docker

Volumes Docker (persistente volume)

```
# docker help volume create
```

Storage Drivers

overlay2
aufs
devicemapper (exemplo: lvm)
btrfs
zfs
vfs

```
LOCAL
SSHFS
NFS
GLUSTERFS
```

Alguns macetes ;)
Docker

Docker run com NFS

```
docker run -d \
    --mount
'type=volume,source=nfsvolume,target=/app,volume-driver=local,volume-opt=type
=nfs,volume-opt=device=:/var/docker-nfs,"volume-opt=o=10.0.0.10,rw,nfsvers=4,
async"' \
    nginx:latest
```

Alguns macetes ;)
Docker

Copiando arquivos entre o seu container e seu linux.

```
docker exec -it my-apache-app bash
…
# exit
docker cp dados.csv my-apache-app:/usr/local/apache2/htdocs/
```

Exportando todos os arquivos de um container para um tarball.

```
docker export my-apache-app -o /root/myapachefiles.tar
```

Alguns macetes ;)
Docker

Criando sua imagem a partir de um container.

```
docker commit -p f51d7285ec41 backup00
docker images
```

Salvando uma imagem para transportar

```
docker save -o backup00.tar backup00
```

Setando uma tag para sua imagem (versão)

```
docker tag backup00 localhost/backup-image:v1
docker images
```

Caso queira enviar para o seu repositorio em hub.docker.com (ou image registry)

```
docker push backup-image:v1
```

Fazendo o load de uma imagem via tar

```
docker load -i /root/backup00.tar
```

Extra Docker

Procurar uma imagem no docker hub
docker search <termo>

Estatística de utilização de recursos do ambiente (host)
`docker stats`

Visualizar todos os processos que um container está executando:
docker top <container>

Utilização de recurso de disco
docker system df

Extra Docker

Configuração do ambiente do seu container:

`/var/lib/docker/containers/[hash container]`

`hostconfig.json`

O docker engine que é responsável por ler e reconfigurar o ambiente, então não basta alterar o arquivo e fazer um stop e start do seu container.

`systemctl restart docker`

Extra Docker

```
docker update (memory / cpu / restart policy)

docker run --cpu-* ... --memory-* ...
```

https://docs.docker.com/engine/reference/commandline/update/

```
docker restart <container>

docker run --restart=always ...
```

https://docs.docker.com/engine/reference/run/#restart-policies---restart

Extra Docker

Persistência de volume

```
$ docker volume create hello

$ docker run -d -v hello:/world busybox ls /world
```

```
$ docker volume create --driver local \
                  --opt type=tmpfs \
                  --opt device=tmpfs \
                  --opt o=size=100m,uid=1000 foovol
```

```
docker volume create meuvol

docker volume ls

docker volume inspect meuvol

docker volume prune

docker volume rm meuvol
```

https://docs.docker.com/engine/reference/commandline/volume_create/

Extra Docker

ENV

Variável de ambiente:

Muito utilizado pelas aplicações para realizar a configuração do ambiente: (nome de usuário, senha …)

```
$ docker run -e KEYCLOAK_USER=<USERNAME> -e KEYCLOAK_PASSWORD=<PASSWORD> jboss/keycloak
```

Build your image - Dockerfile
Docker

Dockerfile

Agora é o momento de criar nossas imagens customizadas (custom) e, para isso, utilizaremos Dockerfile, um arquivo de texto que contém todos os comandos que escrevemos manualmente quando criamos nossos ambientes, só que agora vamos utilizá-lo para construir uma imagem. O Docker cria as imagens lendo as instruções definidas nos arquivos Dockerfile. Alguns comandos do Dockerfile, para mais detalhes podem ser consultados na documentação de Dockerfile.

```
FROM
MAINTAINER
ADD
COPY
ENV
EXPOSE
LABEL
USER
WORKDIR
VOLUME
STOPSIGNAL
ENTRYPOINT
RUN (executa na hora da criação do build)
```

Todo Dockerfile começa definindo qual é a imagem que vai ser utilizada como base.
https://docs.docker.com/engine/reference/builder/

Build your image - Dockerfile
Docker

CMD

A instrução CMD permite definir um comando padrão, que será executado apenas quando você executar o contêiner sem especificar um comando como argumento no docker run. Se o contêiner do Docker for executado com um comando no argumento, o comando padrão será ignorado. Se o Dockerfile tiver mais de uma instrução CMD, todas as instruções, exceto a última, serão ignoradas.

ENTRYPOINT

Já a instrução ENTRYPOINT permite configurar um contêiner que será executado como um executável. É semelhante ao CMD, porque também permite especificar um comando com parâmetros. A diferença é o comando ENTRYPOINT e os parâmetros não são ignorados quando o contêiner do Docker é executado com os parâmetros da linha de comando.

* Existe uma maneira de ignorar ENTRYPOINT, mas é improvável que você o faça.

Build your image - Dockerfile
Docker

Dockerfile (build) sua imagem (zero)

```
root@host:~/onlybash# ls
bin  Dockerfile  lib  lib64

root@host:~/onlybash# cat Dockerfile
FROM scratch
ADD /bin /bin
ADD /lib /lib
ADD /lib64 /lib64
CMD /bin/bash

root@host:~/onlybash# docker build -t onlybash .
```

Build your image - Dockerfile
Docker

Exemplo de um Dockerfile (baseado de uma imagem):

```
FROM node:5.2
RUN npm install bower -g
RUN npm cache clear
ADD ./docker-entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
CMD [ "bash" ]

# docker build -t image_name /home/william/caminhodockerfile/
```

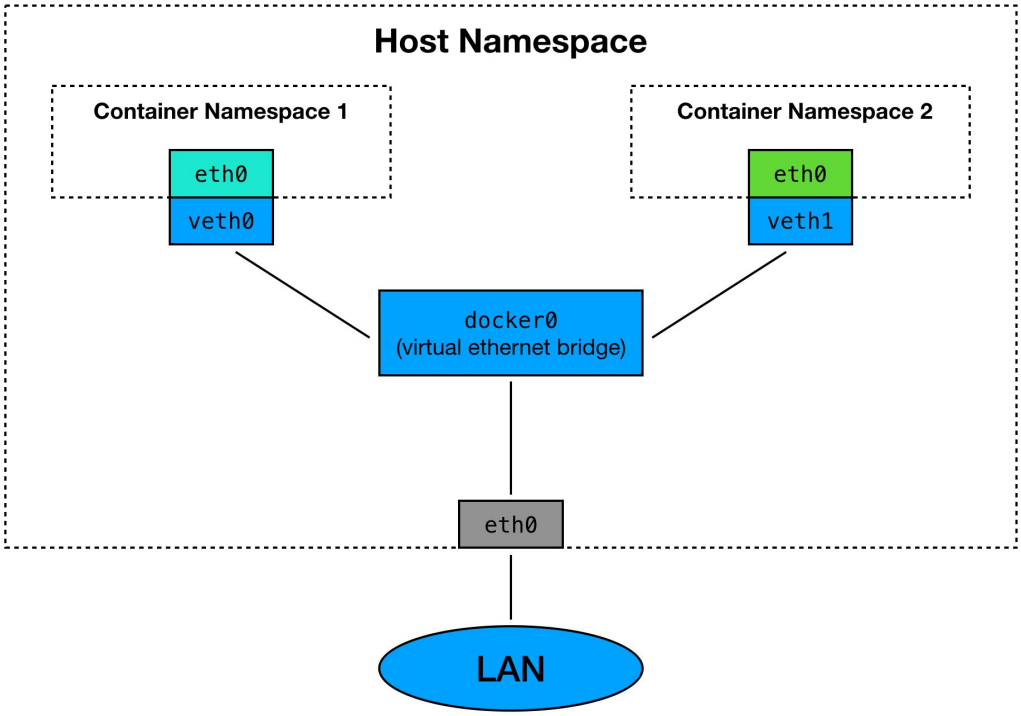
Build your image - Dockerfile
Docker

Exemplo de um Dockerfile (baseado de uma imagem):

```
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/

# docker build -t image_name /home/william/caminhodockerfile/
```

Network
Docker



Portas do container
Docker

Network (portas do container)

Informa a porta que está exposta no container (informativo)

```
# docker run -i --expose=22 b5593e60c33b bash
```

Mapeia uma porta do IP externo do Host com a porta do container (docker-proxy).

```
# docker run -d -p 9080:80 --name http1 httpd
```

Subindo o container na pilha default da rede do Host.

```
# docker run -d --net=host myvnc
```

Tipos de redes
Docker

Network (padrão criar um bridge local)

```
root@virt-01:~# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242835ab07a       no              veth3f57c33
```

```
7: veth3f57c33@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
docker0 state UP group default
    link/ether 22:f8:94:1f:67:48 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::20f8:94ff:fe1f:6748/64 scope link
       valid_lft forever preferred_lft forever
```

Outros tipos de redes:

bridge : bridge local linux
host: mesma interface do host
ipvlan: redes com tag vlan
macvlan: SRV-IO
null: sem rede
overlay: (vxlan) utilizado sobre diversos nós docker.

```
# docker help network create
```

Manipulando redes
Docker

Network

Criar uma nova rede (nettest)

```
# docker network create nettest
```

Iniciando um novo container Docker com a rede criada

```
# docker run --name meucontainerapp -d --network nettest httpd
```

Pegando o endereço IP do container Docker

```
# docker inspect meucontainerapp | grep IPAdd
```

*** Uma nova rede criar uma estrutura de stack para os contêineres sobre.

Artigo sobre tipos de redes:
https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/

Macetes ;)
Docker

Dois container utilizando a mesma pilha de rede (ideal para deploy de DB + APP).

```
# docker run -id -p 80:80 --name api-php-apache php:apache
# docker run -it --net=container:api-php-apache mariadb
```

Criando link entre dois ou mais container : /etc/hosts

```
# docker run -id --name mydb mariadb
# docker run -id --name appweb --link mydb appweb
# docker exec -it appweb bash
curl mydb … OK
```

Docker netns (namespace do container)

Acessando pelo Host a pilha de rede do container

- Pegue um PID de algum processo que esteja rodando no container (pilha netns).

```
# ps -aux | grep processo
```

- Crie diretório temporário do netns (estrutura de leitura do comando ip netns)

```
# mkdir /var/run/netns
```

- Crie um link simbólico do estado do processo (ns/net)

```
# ln -sf /proc/16426/ns/net /var/run/netns/pilhacontainerdocker
```

- Acesse pilha de rede do container docker pelo o Host com o comando

```
# ip netns exec pilhacontainerdocker /bin/bash
```

Extra Docker

```
docker run -p 80:80 ….

# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      691/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1099/sshd
tcp6       0      0 :::80                   :::*                    LISTEN      19448/docker-proxy
tcp6       0      0 :::22                   :::*                    LISTEN      1099/sshd

# iptables -L -n -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
MASQUERADE  tcp  --  172.17.0.2           172.17.0.2           tcp dpt:80

Chain DOCKER (2 references)
target     prot opt source               destination
RETURN     all  --  0.0.0.0/0            0.0.0.0/0
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:80 to:172.17.0.2:80
```

https://windsock.io/the-docker-proxy/

Extra Docker

```
# docker-proxy --help

Usage of /usr/bin/docker-proxy:

  -container-ip string
        container ip
  -container-port int
        container port (default -1)
  -host-ip string
        host ip
  -host-port int
        host port (default -1)
  -proto string
        proxy protocol (default "tcp")
```

Extra Docker

Network settings

```
--dns=[]            : Set custom dns servers for the container
--network="bridge" : Connect a container to a network
                      'bridge': create a network stack on the default Docker bridge
                      'none': no networking
                      'container:<name|id>': reuse another container's network stack
                      'host': use the Docker host network stack
                      '<network-name>|<network-id>': connect to a user-defined network
--network-alias=[] : Add network-scoped alias for the container
--add-host=""       : Add a line to /etc/hosts (host:IP)
--mac-address=""    : Sets the container's Ethernet device's MAC address
--ip=""             : Sets the container's Ethernet device's IPv4 address
--ip6=""            : Sets the container's Ethernet device's IPv6 address
```

```
 --link-local-ip=[] : Sets one or more container's Ethernet device's link local IPv4/IPv6 addresses
```

https://docs.docker.com/engine/reference/run/#network-settings

Extra Docker

docker run --link …

DNS

docker network create projeto

docker run itd --net projeto --name app1 …
docker run itd --net projeto --name app2 …
docker run itd --net projeto --name app3.sub …

app1.projeto / app2.projeto / app3.sub.projeto

** netns e iptables dockerd (dns)

Extra Docker

`docker network connect …`

https://docs.docker.com/engine/reference/commandline/network_connect/

Compose
Docker

Docker Compose - docker-compose.yml

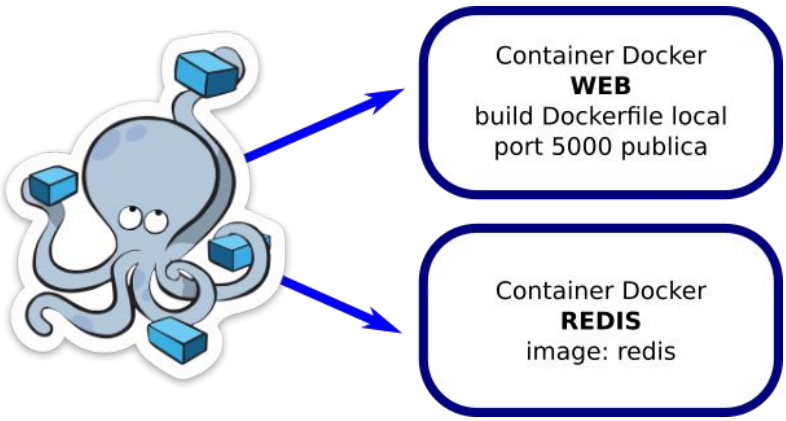
```
version: '3'
Services:

  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis

  redis:
    image: redis

volumes:
  Logvolume01: {}

# docker-compose up
```



https://docs.docker.com/compose/

```
version: '3'

services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
```

Docker Ferramentas

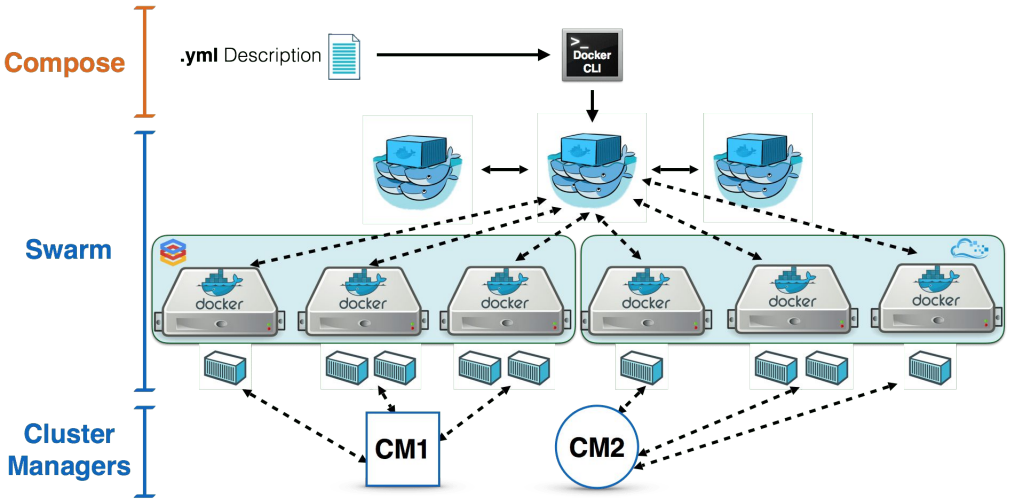
- https://portainer.io/

```
$ docker volume create portainer_data
$ docker run -d -p 8000:8000 -p 9000:9000 \
--name=portainer --restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data portainer/portainer
```

- https://cockpit-project.org/ (plugin para Docker)
- https://kitematic.com/ (MacOS e Windows)

Ferramentas para cluster
Docker

Ferramentas de Gerenciamento para Docker



https://blog.docker.com/2015/11/deploy-manage-cluster-docker-swarm/

https://docs.docker.com/engine/swarm/

Clusters
Docker

Ferramentas de Gerenciamento para Docker (Orquestrador)

https://www.docker.com/kubernetes