DOCKER

[Curso Udemy]

O QUE É DOCKER

É uma plataforma que foi criada para construir, rodar e transferir aplicações do seu ambiente de desenvolvimento para o ambiente de produção.

Com o Docker, construímos **containers.** No container, está toda a nossa aplicação, sendo elas o projeto, dependências e pacotes.

A vantagem do container é manter o projeto funcional em qualquer máquina que tenha Docker, pois dentro de um container, mantemos as versões do projeto, dependência e configurações de ambiente.

Outra vantagem seria que, as versões de programas dentro de um container não entram em conflito com as versões do mesmo programa instalado no seu sistema operacional.

DIFERENÇA DE VM's E CONTAINERS

Hypervisor:

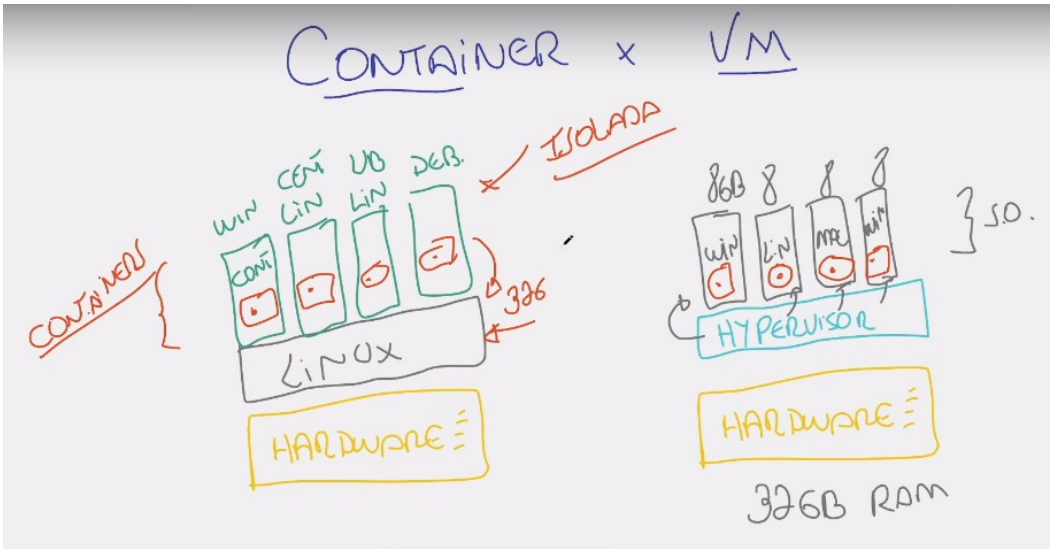
Te dá a possibilidade de virtualizar outros sistemas operacionais. Um exemplo de hypervisor seria o **virtualbox.**

A desvantagem do hypervisor é que ele utiliza o hardware (pc) como seu limitador. **Um exemplo seria:** Um computador tem 16gb de ram. Com esses 16gb, virtualizo 2 SO com 8gb de ram cada, feito isso, não consigo criar mais virtualizações, pois cada virtualização gasta memória, processamento e CPU do meu hardware.

Containers:

Te dá a possibilidade de executar cada aplicação de forma isolada, consumindo menos recursos do hardware e conseguimos iniciar a aplicação de uma maneira mais rápida. O Container é executado diretamente no sistema operacional do seu computador, com isso, conseguimos criar e trabalhar com vários containers de uma só vez.

Dentro de cada container, podemos inserir a imagem do SO que precisamos trabalhar (sendo ela Windows ou várias versões de Linux), criar nossa aplicação e instalação de dependências de acordo com a nosso projeto.



INSTALAÇÃO DO DOCKER NO LINUX:

1º - Instalar Docker Engine

2º - Instalar o Docker Desktop

Para descobrir as versões do Docker após instalação:

$ docker compose version
Docker Compose version v2.17.3

$ docker –version
Docker version 23.0.5

$docker version
Client: Docker Engine – Community
Cloud integration: v1.0.31
Version: 23.0.5
API version: 1.42

COMO FUNCIONA O DOCKER

Dentro da nossa aplicação, criamos um arquivo chamado **Dockerfile.** Dentro dele, colocamos os parâmetros para que a imagem possa ser criada.

O Dockerfile é o responsável por tirar a aplicação e criar uma imagem. A criação da imagem é de acordo com os parâmetros colocados dentro do Dockerfile.

Dentro da imagem, temos:

→ Um pedaço do SO
→ Ambiente (exemplo: NODE)
→ Arquivos
→ Bibliotecas
→ Variáveis

Após a criação da imagem, podemos colocá-la dentro de um container. Com isso, podemos fazer a transferência de ambiente de desenvolvimento para produção.

Dentro do meu container criado, temos a nossa imagem, o SO, as dependências e nossa aplicação.

COMO É O DOCKER HUB?

Docker Hub

O Docker Hub é um repositório de containers. Nele é onde conseguimos colocar a nossa **imagem.** O Docker Hub é a maior biblioteca para a comunidade de containers e imagens.

CRIANDO A PRIMEIRA IMAGEM COM NODEJS

Observação: Dentro do Dockerfile, criamos as instruções para criar a imagem

Passo a passo:

→ Criar um arquivo .js e colocar um script (hello world)
→ Habilitar a extensão Docker no VSCode
→ Criar um arquivo Dockerfile na raiz do projeto e dentro dele:

```
FROM node :alpine
COPY . /app
WORKDIR /app
CMD node 01-app.js

// FROM - Utilizaremos node com SO alpine
// COPY - Copia tudo que está dentro do diretório atual para o novo diretório (imagem)
// WORKDIR – Trabalhar sempre dentro desse diretório
// CMD - Vai utilizar o CMD para rodar a aplicação
```

→ No terminal, na pasta raiz:
// Utilizado para criar a imagem a partir do diretório local (**.** = diretório local)

```
docker build -t ola-mundo .
```

→ Para listar as imagens do Docker:

```
docker images
```

PARA RODAR A IMAGEM, CRIAMOS O CONTAINER:

Podemos criar um container rodando a imagem **diretamente** no software do Docker. Para rodar no terminal, basta:

→ No terminal

```
docker run ola-mundo
```

COMANDOS BÁSICOS DO DOCKER

→ **docker ps** // lista os containers que estão funcionando
→ **docker ps -a** // lista os containers que possuo na máquina
→ **docker pull nomeImagem** // baixar a imagem
→ **docker run -it nomeImagem** //baixa a imagem de forma interativa
→ **docker images** //visualiza as imagens criadas
→ **docker build -t app .** //builda a aplicação para uma imagem
→ **docker –help** // Colocando –help no final, você tem informações de como aquele comando funciona
→ **docker run -d –name manga -p 3000:3000 app:v2** //Inicia o container em background , renomeia o nome do container e inicia ele na porta 3000

Comandos mais executados

LINUX

Distribuição do linux:

→ Ubuntu
→ Debian
→ CentOS
→ Fedora
→ Alpine

Quando utilizamos uma imagem do Ubuntu, ela vem sem nenhum aplicativo, portanto, devemos instalá-los, os aplicativos na imagem do Ubuntu se chama **packages.**

PASTAS LINUX:

Estruturas de diretórios

/ → Pasta raiz
/bin → Contem binários essenciais para o usuário
/boot → Responsável pela inicialização do SO
/cdrom → Utilizado para ler CDs e DVDs colocados no Ubuntu
/dev → Responsável por ler os arquivos de dispositivos
/etc → Arquivos de configurações diversas
/home → Diretório do usuário
/lib → Biblioteca compartilhadas essenciais do sistema
/lost+found → Arquivos recuperados
/media → Monta CDs e DVDs inseridos
/mnt → Pontos de montagem temporários
/opt → Pacotes opcionais
/proc → Arquivos de processos e kernel
/root → Diretório padrão do usuário root
/run → Arquivos de estado de aplicações
/srv → Dados de serviços
/sys → Informações sobre o sistema de hardware
/tmp → Arquivos temporários
/usr → Binário dos usuários
/var → Arquivos de dados variáveis

ATALHOS:

Para instalar um package, basta:

apt install nano

Para remover um package, basta:

apt remove nano

→ GREP

Utilizado para procurar palavras dentro de um arquivos.
Exemplo:

grep hello helloWorld.txt

Ou:

grep -i -r hello . //pesquisa em todos os diretórios da pasta atual

→ FIND

Lista todos os diretórios na pasta atual
Exemplo:

find

Ou:

find ./teste

Ou:

find -type f -name "'docker" //retorna por pesquisa do nome

→ EXECUTAR MULTIPLOS COMANDOS

Utilize a virgula para executar vários comandos
Exemplo:

mkdir teste; touch teste.txt; echo HelloWorld > teste.txt ; cat teste.txt

Ou:

mkdir teste && cd teste && echo ok

→ GERENCIAMENTO DE PROCESSOS

Mostra os processos que estão sendo utilizados
Exemplo:

ps

Exemplo:

sleep 5 // Trava o terminal por 5 segundos

Após

sleep 5 & // Executa em background

Após

ps

Para matar um processo:
Pegar a id e utilizar kill
Exemplo:

kill 502

→ COMO CRIAR UM USUÁRIO NO LINUX POR SHELL

No terminal:

useradd -m elbert // para criar um usuário
userdel elbert // para criar um usuário

Para realizar login com o novo usuário:
- Abrir um novo terminal

docker ps // listar os containers

- Copiar o ID do container

Após:

docker exec -it -u elbert 4622bc054db7 bash

→ GERENCIANDO GRUPOS

Para listar os grupos:

cat /etc/group

Para verificar os grupos que o usuário pertence:

groups elbert

Para adicionar um grupo:

groupadd docker

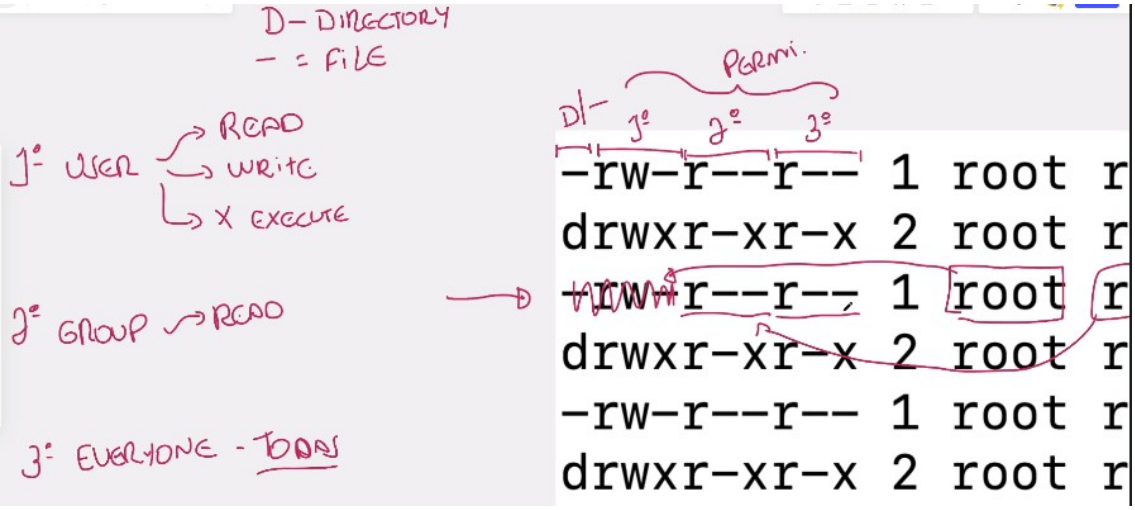
Para verificar os comandos:

usermod

Para inserir um usuário dentro do grupo:

usermod -G docker elbert

→ PERMISSÕES



Para trocar uma permissão:

chmod u+x docker.txt

// **x** = execute, poderia ser **r** ou **w**.

CRIANDO IMAGENS DOCKER

→ O QUE É UMA IMAGEM

Uma imagem contém tudo que é necessário para a sua aplicação funcionar

Dentro de uma imagem, temos:

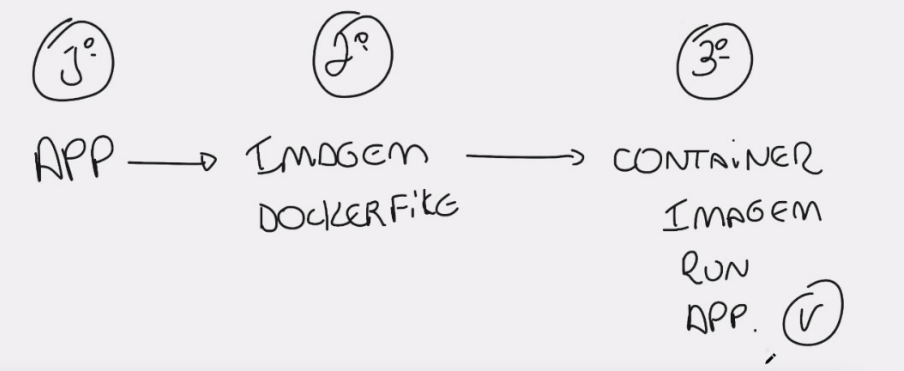
→ Sistema operacional
→ Bibliotecas
→ Arquivos da aplicação
→ Variáveis de ambiente

→ O QUE É UM CONTAINER

É um processo que roda, carrega a imagem e virtualiza em um container de forma isolada

Dentro do container, temos

→ Um ambiente(virtualização) isolado
→ start/stop (Posso parar e startar um container)
→ Processo que roda dentro de uma máquina



1º – Criamos a nossa aplicação
2º - Convertemos ela para uma imagem
3º – Se cria um container, ele é criado a partir da imagem aonde ele executa a aplicação.

DOCKERFILE

→ **FROM :** Qual imagem vai carregar/plataforma
→ **WORDKDIR:** Trabalhar no diretório
→ **COPY /**
→ **ADD :** Utilizado para copiar arquivos de endereços web. Ele também descompacta arquivos .zip do seu local
→ **RUN:** Utilizado quando está construindo a imagem
→ **ENV:** Utilizado para consumir uma API
→ **EXPOSE:** Configura a porta do localhost
→ **USER:** Qual usuário vai executar a aplicação
→ **CMD:** Rodar comandos CMD, depois de ter criado a imagem (**RUN**)
→ **ENTRYPOINT:** Rodar comandos

CRIANDO A APLICAÇÃO ATRAVÉS DOCKER SAMPLE APPLICATION

Utilizar o projeto

```
git clone https://github.com/docker/getting-started.git
```

Criar um Dockerfile e colocar o seguinte atributo:

```
FROM node:12-alpine
```

No terminal:

docker build -t app .

Para ver as imagens criadas:

docker images

Para criar um container:

docker run -it app sh //Cria um container e executa no shell

Voltando ao Dockerfile:

WORKDIR /app //A partir dos comandos abaixo, todos estarão sendo executados dentro do diretório /app

COPY . . //Copia todos os arquivos para /app

Para atualizar a imagem:

docker build -t app .

Para criar um container:

docker run -it app sh

Voltando para o Dockerfile, instalar as dependências do projeto

RUN apk add --no-cache python2 g++ make

Ainda no Dockerfile: (configurar a porta do localhost)

EXPOSE 3000

Para criar um usuário e dar permissão para ele (apenas exemplo)

```
3 RUN addgroup dev && adduser -S -G andre dev
4 USER andre
```

Depois, criar o CMD com os comandos para iniciar o node

CMD [“node”, “src/index.js”]

Ao final, o escopo do Dockerfile vai ficar :

```
Dockerfile > ...
1  FROM node:12-alpine
2  WORKDIR /app
3  COPY . .
4  RUN apk add --no-cache python2 g++ make
5  RUN yarn install --production
6  CMD ["node", "src/index.js"]
7  EXPOSE 3000
```

Para executar o container:

docker run -p 3000:3000 minha_imagem

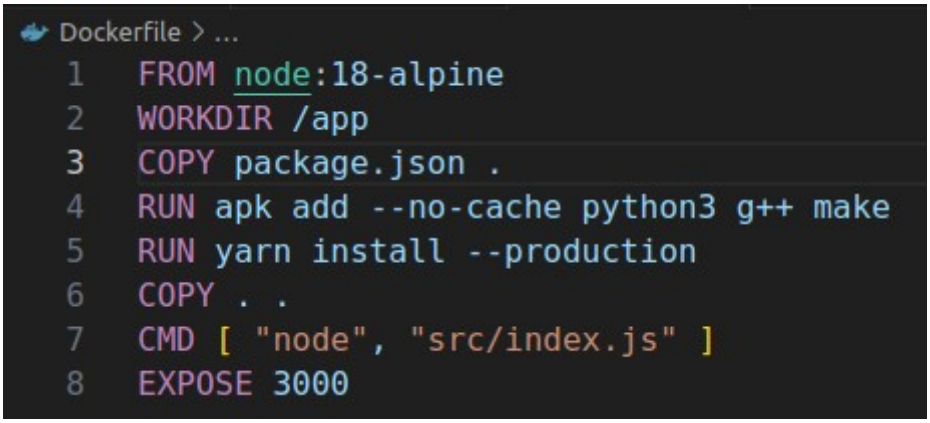
// **-p** = permite colocar qual é a porta do host e a do container

// **-dp** = roda em background

PARA OTIMIZAR O BUILD DA IMAGEM:

Utilizar o COPY antes da instalação das dependências para verificar se elas já estão instaladas, caso estejam, é utilizado o cache das dependências e pula a etapa de instalação do RUN e parte para o COPY dos arquivos.

Caso seja a primeira instalação, ele executa a instalação das dependências.



COMO MUDAR A TAG DE UMA IMAGEM:

Para evitar conflitos de nomes de imagens iguais, é necessário trocar a tag da imagem.

Para criar uma iamgem com as tags, basta:

docker build -t app:v1.0.0 .

Para remover a imagem:

docker image remove app:v1.0.0

Para renomear a tag da imagem e criar uma copia da mesma:

docker image tag app:latest app:v1.0.0

COMO COMPARTILHAR UMA IMAGEM:

Passo a passo:

→ Criar um repositório no Docker hub

→ Colocar o nome do repositório com o mesmo nome da imagem

Criar uma tag da imagem id com o nome do repositório

docker image tag a65c257c8528 elbertjean/app:v1

Após, fazer o Docker login

docker login

Para realizar o upload da imagem:

docker push elbertjean/app:v1

O primeiro **PUSH** será mais demorado, pois a imagem ainda não existe no repositório. Nas demais versões que utilizarmos o push, será mais rápida, pois só será atualizado o que foi modificado, criando assim, uma nova versão da anterior.

COMO SALVAR E CARREGAR AS IMAGENS LOCAL:

Para salvar:

Para saber as opções de salvar

docker image save --help

Para realizar o salvamento em formato .tar (comprimido). Esse arquivo é compactado na pasta atual

docker image save -o appv2.tar app:v1

Para saber as opções de restaurar

docker image load –help

Para restaurar (estar na pasta do .tar comprimido)

docker image load -i appv2.tar

CONTAINERS

Ele te dá a possibilidade de executar cada aplicação de forma isolada, consumindo menos recursos do hardware e conseguimos iniciar a aplicação de uma maneira mais rápida. O Container é executado diretamente no sistema operacional do seu computador, com isso, conseguimos criar e trabalhar com vários containers de uma só vez.

Para listar os containers:

docker ps -a // sem o **-a**, ele mostra os containers em execução

Para iniciar um container

docker run app: v2

Para inserir um nome no container **(!IMPORTANTE)**

docker run -d –name manga -p 3000:3000 app:v2

PARA VERIFICAR OS LOGS DOS CONTAINERS

Verificar os logs sempre quando algo estiver dando errado:

Para saber as opções de salvar
docker logs --help

Para verificar os containers em execução
docker ps

Para mostrar os logs que acontecem
docker logs -f nomeContainer

Para mostrar as últimas 10 linhas
docker logs -n nomeContainer

Para mostrar todo o log:
docker logs nomeContainer

Para mostrar o tempo do continar
docker logs -t nomeContainer

PORTAS VIRTUAIS NO DOCKER

Para mapear as portas do container com o SO, basta:
docker run -d -p local:container nomeContainer

Assim:
docker run -d -p 80:3000 nomeContainer

Ou então:
docker run -d -p 80:3000 nomeContainer

EXECUTANDO COMANDOS EM CONTAINERS

Pesquisar pelos arquivos de um container sem precisar entrar nele
docker exec bananinha ls

INICIANDO E PARANDO CONTAINERS

Comando executados para inicir e parar um container

Para iniciar:
docker start nomeContainer

Para parar:
docker stop nomeContainer

REMOVENDO CONTAINERS

Para remover um container:

docker remove nomeContainer

VOLUMES PERSISTENTES

Nada mais é do que fazer o backup de uma pasta do container apagado. Volumes são uma forma de persistir dados em aplicações e não depender de containers para isso

Para criar um volume

docker volume create app-dados

Para inspecionar o volume:

docker volume inspect app-dados

Para inserir um container em um volume:

docker run -dp 3000:3000 --name kiwi -v app-dados:/app/dados app:v1

-v = Volume

/app/dados = Local onde meus arquivos são gravados

Para verificar se a aplicação ocorreu certo

docker exec -it kiwi sh

COPIAR ARQUIVOS DO CONTAINER PARA O LOCAL

Para isso, basta abrir o terminal e:

docker cp origem destino

Com isso, para moverdo container para o local:

docker cp kiwi2:/app/teste.txt . // . = pasta local

Para fazer o caminho inverso:

docker cp elbert.txt kiwi2:/app

DOCKER COMPOSE

Responsável por fazer a distribuição da aplicação, conseguindo separar a aplicação de frontend, backend e banco de dados e ao final, juntar todos como uma só aplicação

Para verificar a versão do docker compose:

docker compose version

Antes de iniciar os estudos de docker compose, precisamos apagar todas os containers, imagens e volumes do docker, para isso, basta ir em:

Docker Desktop/troubleshoot/ clean/purge data

Realizando testes no projeto netflix:

Para executar o projeto:

docker compose up //para construir o docker

ARQUIVO DOCKER-COMPOSE

Dentro do arquivo docker-compose, temos as configurações do ele deve executar, quantos containers, quais container executar, quais os nomes, portas, configurações e afins.

YAML .yml (IAMOL): Ele consegue passar instruções para algum programa ou aplicativo conseguir ler essas instruções de cima para baixo e em um sequência lógica. **Resumindo, o YAML é uma linguagem legível de serialização de dados usada na escrita de arquivos de configuração**

O projeto de organização do YAML se chama **indentação.**

docker-compose.yml

```yaml
version: "3.8"                  #identifica a versão do docker-compose

services:
  frontend:                     #nome dos containers
    depends_on:                 #de quem essa aplicação depende?
      - backend
    build: ./frontend           #constroi esse build de acordo com o diretório do Dockerfile do frontend
    ports:                      #definimos as portas de acordo com o Dockerfile
      - 3000:3000

  backend:
    depends_on:
      - db
    build: ./backend
    ports:
      - 3001:3001
    environment:                #faz a comunicação com o banco de dados
      DB_URL: mongodb://db/vidly
    command: ./docker-entrypoint.sh

  db:
    image: mongo:4.0-xenial     #definimos a imagem do banco de dados
    ports:
      - 27017:27017             #porta padrão do mongo
    volumes:                    #nesta aplicação, se utiliza o volume vidly
      - vidly:/data/db          #armazena dentro de db, senão, é armazenada na parte volátil

volumes:
  vidly:    #preciso do volume vidly por conta da aplicação
```

PARA INICIAR O DOCKER COMPOSE:

docker compose up

Para saber as opções do docker compose

docker compose –help

Para buildar o docker compose:

docker compose up –build

Após o primeiro build, basta:
(Lembrando de rodar na pasta local do docker-compose.yml)

docker compose run -d

Para parar a aplicação, basta:

docker compose down

DOCKER NETWORK

COMO OS CONTAINERS SE COMUNICAM COM DOCKER?

O Docker cria uma rede virtual, ele tem um servidor dns próprio, onde ele tem um dns resolver onde ele consegue mapear o host name do container com o endereçamento ip

PARA VERIFICAR ERROS NO DOCKER COMPOSE

Basta:

docker compose logs

Docker Cheat Sheet

docker

Build

Build an image from the Dockerfile in the current directory and tag the image
`docker build -t myimage:1.0 .`

List all images that are locally stored with the Docker Engine
`docker image ls`

Delete an image from the local image store
`docker image rm alpine:3.4`

Share

Pull an image from a registry
`docker pull myimage:1.0`

Retag a local image with a new image name and tag
`docker tag myimage:1.0 myrepo/myimage:2.0`

Push an image to a registry
`docker push myrepo/myimage:2.0`

Run

Run a container from the Alpine version 3.9 image, name the running container "web" and expose port 5000 externally, mapped to port 80 inside the container.
`docker container run --name web -p 5000:80 alpine:3.9`

Stop a running container through SIGTERM
`docker container stop web`

Stop a running container through SIGKILL
`docker container kill web`

List the networks
`docker network ls`

List the running containers (add `--all` to include stopped containers)
`docker container ls`

Delete all running and stopped containers
`docker container rm -f $(docker ps -aq)`

Print the last 100 lines of a container's logs
`docker container logs --tail 100 web`

Docker Management

All commands below are called as options to the base `docker` command. `Run docker <command> --help` for more information on a particular command.

`app*`	*Docker Application*
`assemble*`	*Framework-aware builds (Docker Enterprise)*
`builder`	*Manage builds*
`cluster`	*Manage Docker clusters (Docker Enterprise)*
`config`	*Manage Docker configs*
`context`	*Manage contexts*
`engine`	*Manage the docker Engine*
`image`	*Manage images*
`network`	*Manage networks*
`node`	*Manage Swarm nodes*
`plugin`	*Manage plugins*
`registry*`	*Manage Docker registries*
`secret`	*Manage Docker secrets*
`service`	*Manage services*
`stack`	*Manage Docker stacks*
`swarm`	*Manage swarm*
`system`	*Manage Docker*
`template*`	*Quickly scaffold services (Docker Enterprise)*
`trust`	*Manage trust on Docker images*
`volume`	*Manage volumes*

*Experimental in Docker Enterprise 3.0.

www.docker.com