Comandos Gerais

- Não rodar como DEAMON quer dizer não rodar em primeiro plano
  PARÂMETRO: -d
- Parâmetro -ti serve para dizer ao docker que queremos interatividade, já subimos conectado no container

Criar uma imagem utilizamos o comando

```
docker container run <NOME>
```

Criar um container sem colocar em execução, diferente do run

```
docker container create
```

Startar um container depois de sair dele

```
docker start -i <CONTAINER ID>
```

Pausar um container

```
docker container stop <CONTAINER ID>
```

Restartar um container

```
docker container restart <CONTAINER ID>
```

Visualizar os containers ativos use o comando

```
docker container ls
```

Visualizar os container mesmo sem estarem ativos

```
docker container ls -a
```

Docker container attach: Conecta no container

- Precisa passar CONTAINER ID

```
docker container attach <CONTAINER ID>
```

Inspect

- —pretty: Visualização mais limpa dos dados

Serve para visualizar informações do container

```
docker container inspect <CONTAINER ID>
```

Apagar container

- -f para forçar a apagar mesmo estando em execução

```
docker container rm -f <CONTAINER ID>
```

Entrar no bash

```
docker container exec -ti <CONTAINER ID > bash
```

Visualizar status de um container

```
docker container stats <CONTAINER ID >
```

Top

- Mostra os processos do container

```
docker container top <CONTAINER ID>
```

Delimitar uso de memória container

```
docker container run -d -m <QUANTIDADE DE MEMORIA>M <CONTAINER ID>
```

Delimitar uso de CPU do container

```
docker container -run -d --cpus <QNT CORES> <CONTAINER ID >
```

Upgrade de memoria ou CPU em um container

```
docker container update --cpus <QNTD CORES> <CONTAINER ID>
```

Commit

- Transforma CONTAINER em IMAGEM

```
docker commit -m "DESCRIPTION" <CONTAINER ID>
```

Dockerfile

```
FROM debian

LABEL app="Giropops"
ENV IGOR="Owner"

RUN apt-get update && apt-get install -y stress && apt-get clean

CMD stress --cpu 1 --vm-bytes 64M --vm 1
```

Build

- “Construir a imagem que criamos no DOCKERFILE”
- . caso você esteja na pasta do Dockerfile

- Sempre colocar depois do nome do container sua versão: Teste:2.0

```
docker image build -t <CONTAINER ID> .
```

Sair do conteiner encerrando a sessão: CTRL + D/Logout/Exit

Sair do conteiner sem encerrar ele: CTRL + P + Q

Volume/Bind/Prune

Volumes

Bind

- Se um container morrer com um conteudo, o conteudo criado ficará salvo no diretorio apontado
- bind = Quando já tenho um diretório e quero montar ele dentro do container
- Parametro -ro = Read Only

```
docker container run -ti --mount type=bind,src=<DIRETORIO>,dst=/srcfinal <imagem>
```

Volume

- Todos os volumes criados ficam no diretório: /var/lib/docker/volumes/
  - Lista todos os volumes

```
docker volume ls
```

  - Cria um volume

```
docker volume create <nome_do_volume>
```

Criar container com tipo volume

- dst = destination

```
docker container run -ti --mount type=volume,src=giropops,dst=/giropops debian
```

Prune

- Remove containers, volumes, imagens que não estão sendo utilizadas

```
docker container prune / docker volume prune / docker image prune
```

Backup

```
docker container run -ti --mount type=volume,src=desafio,dst=/data

--mount type=bind,src=/opt/backup,dst=/backup debian tar -cvf /backup/bkp_bando.tar /data
```

Volumes: Comandos

Dockerfiles

Estrutura de um Dockerfile

```
ADD => Copia novos arquivos, diretórios, arquivos TAR ou arquivos remotos e os adicionam ao filesystem do co
ntainer;

CMD => Executa um comando, diferente do RUN que executa o comando no momento em que está "buildando" a image
m, o CMD executa no início da execução do container;

LABEL => Adiciona metadados a imagem como versão, descrição e fabricante;

COPY => Copia novos arquivos e diretórios e os adicionam ao filesystem do container;

ENTRYPOINT => Permite você configurar um container para rodar um executável, e quando esse executável for fi
nalizado, o container também será;

ENV => Informa variáveis de ambiente ao container;

EXPOSE => Informa qual porta o container estará ouvindo;

FROM => Indica qual imagem será utilizada como base, ela precisa ser a primeira linha do Dockerfile;

MAINTAINER => Autor da imagem;

RUN => Executa qualquer comando em uma nova camada no topo da imagem e "commita" as alterações. Essas altera
ções você poderá utilizar nas próximas instruções de seu Dockerfile;

USER => Determina qual o usuário será utilizado na imagem. Por default é o root;

VOLUME => Permite a criação de um ponto de montagem no container;

WORKDIR => Responsável por mudar do diretório / (raiz) para o especificado nele;
```

```
FROM debian

RUN apt-get update && apt-get install -y apache2 && apt-get clean
RUN chown www-data:www-data /var/lock && chown www-data:www-data /var/run/ && chown www-data:www-data /var/log/

ENV APACHE_LOCK_DIR="/var/lock"
ENV APACHE_PID_FILE="/var/run/apache2.pid"
ENV APACHE_RUN_USER="www-data"
ENV APACHE_RUN_GROUP="www-data"
ENV APACHE_LOG_DIR="/var/log/apache2"

ADD index.html /var/www/html

LABEL description="Webserver"
LABEL version="2.0.0"

WORKDIR /var/www/html/

VOLUME /var/www/html/
EXPOSE 80

ENTRYPOINT ["/usr/sbin/apachectl"]
CMD ["-D", "FOREGROUND"]
```

Exemplo 2

```
FROM debian

RUN apt-get update && apt-get install -y apache2 && apt-get clean
ENV APACHE_LOCK_DIR="/var/lock"
ENV APACHE_PID_FILE="/var/run/apache2.pid"
ENV APACHE_RUN_USER="www-data"
ENV APACHE_RUN_GROUP="www-data"
ENV APACHE_LOG_DIR="/var/log/apache2"

LABEL description="Webserver"

VOLUME /var/www/html/
EXPOSE 80
```

Exemplo 3

```
FROM debian

RUN apt-get update && apt-get install -y apache2 && apt-get clean
ENV APACHE_LOCK_DIR="/var/lock"
ENV APACHE_PID_FILE="/var/run/apache2/apache2.pid"
ENV APACHE_RUN_USER="www-data"
ENV APACHE_RUN_DIR="/var/run/apache2"
ENV APACHE_RUN_GROUP="www-data"
ENV APACHE_LOG_DIR="/var/log/apache2"

LABEL description="Webserver"

VOLUME /var/www/html/
EXPOSE 80

ENTRYPOINT ["/usr/sbin/apachectl"]
CMD ["-D", "FOREGROUND"]
```

Exemplo 4

```
FROM golang

WORKDIR /app
ADD . /app
RUN go build -o goapp
ENTRYPOINT ./goapp


FROM golang AS buildando

ADD . /src
WORKDIR /src
RUN go build -o goapp


FROM alpine:3.1

WORKDIR /app
COPY --from=buildando /src/goapp /app
ENTRYPOINT ./goapp
```

-P = Procura se existe uma porta exposta

-p = Deixa setar uma porta para inciar o container

```
docker container run -ti -P
```

Build

- “Construir a imagem que criamos no DOCKERFILE”
- . caso você esteja na pasta do Dockerfile
- Sempre colocar depois do nome do container sua versão: Teste:2.0

```
docker image build -t <CONTAINER ID> .
```

```
docker container run -d -p 8080:80 apache:2.0.0
```

Docker Swarm

Service

Workers:

- Responsável por ter containers em execução

Manager:

- Responsável pela administração dos containers
- Informaçãoes dos containers ficam no manager

- 3 nodes e os 3 forem MANAGERS caso 1 deles caia o cluster continuará funcionando
- Preciso ter 51% do meu cluster funcionando para o Docker Swarm funcionar

---

Iniciar Docker Swarm

```
docker swarm init
```

Entrar no cluster

```
docker swarm join --token <TOKEN GERADO PELO INIT>
```

Promover um node a Manager

```
docker node promote <nome do node>
```

Tirar um node de Manager

```
docker node demote <nome do node>
```

Sair de um cluster

- Só pode usar o LEAVE caso não seja um Manager, caso contrário use leave -f

```
docker swarm leave
```

Exibir o token novamente caso queira entrar com outro node

- Caso queira trocar o token usar parametro —rotate

```
docker swarm join-token {worker} or {manager}
```

Secrets

Service

NFS COMPARTILHAMENTO DE DIRETORIOS >> https://www.linuxtips.io/path-player?courseid=descomplicando-o-docker&unit=634986fed0bfd7f899054a18Unit

---

Criar Serviços

- docker service create --name 'nome do serviço' --replicas 'qnt de serviços' -p 8080:80 'servico'

```
docker service create --name webServer --replicas 3 -p 8080:80 nginx
```

Listar Serviços

- docker service ps ‘nome do serviço’

```
docker service ps webServer
```

Availability

- Pause: Não recebe serviços caso eu de um update
- Active: Volta a receber serviços
- Drain: Migra os serviços do node para outro
- Sintaxe: docker node update —availability ‘status’ ‘nome do node’

```
docker node update --availability pause server1
```

Scale

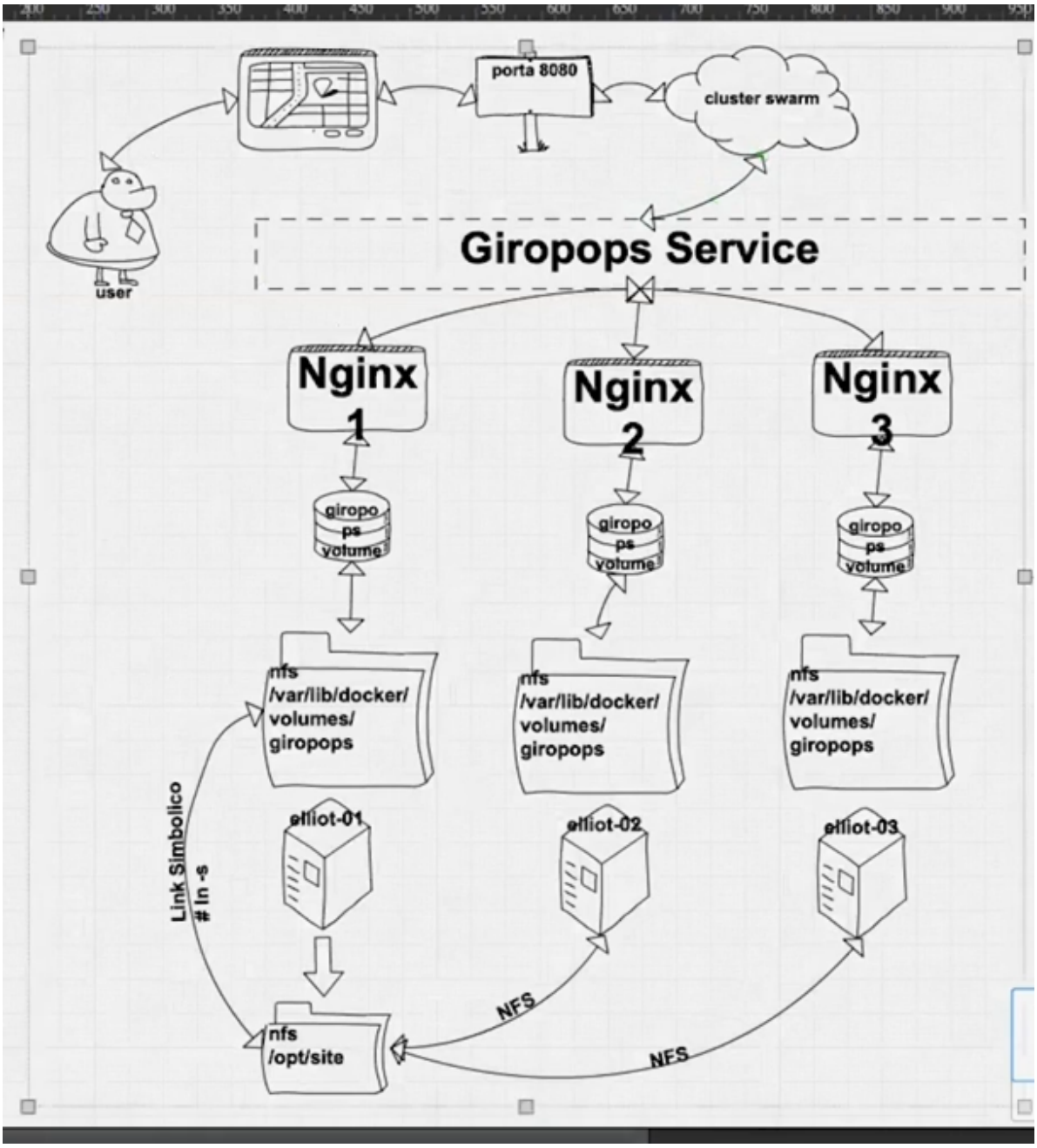
- Se eu utilizar scale em um node com status de pause ele não vai aceitar novos serviços, sendo escalados nos outros nodes
- Sintaxe: docker service scale ‘nome do serviço’=’quantidade de serviços’

```
docker serve scale webServer=10
```

Logs

```
docker service logs -f 'nome do serviço'
```

NFS



Parametros para criação de um serviço

- —hostname
- —env
- —limit-cpu
- —limit-memory

```
docker service create --name giropops --replicas 3 -p 8080:80
--mount type=volume,src=giropops,dst=/usr/share/nginx/html
--hostname nginx-dev --limit-cpu 0.25 --limit-memory 64M --env igor=adm --dns 8.8.8.8 nginx
```

Secrets

Caminho dos secrets nos containers = /run/secrets/

Criar um secret

- Secret Arquivo = docker secrete create <nome da secret> <nome do arquivo>

```
docker secret create igor-arquivo teste.txt
```

Adicionar uma secret ao serviço

- Parâmetro: —secret <nome_do_secret>

```
docker service create --name nginx -p 8080:80 --secret igor-arquivo nginx
```

Update

- Parametro —secret-add / —secret-rm
- Sintaxe: docker serve update —secret-add <nome do secret> nginx

```
docker service update --secret-add igor nginx
```

Secrets especificos

- target > nome
- src > nome do secret
- uid/gid > users
- mode > permissao

```
docker service create --name nginx2 -p 8088:80 --secret
src=igor-arquivo,target=meu-secret,uid=200,gid=200,mode=0400 nginx
```

Docker Compose

docker stack ps <nome da stack>

Criando primeiro compose

```
version: "3.7"
services:
  web:
    image: nginx
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "8080:80"
    networks:
    - webserver
networks:
  webserver:
```

Deployar um compose

- docker stack deploy -c docker-compose.yml <nome do stack>

```
docker stack deploy -c docker-compose.yml giropops
```

Criação de Wordpress e Mysql

```
version: '3.7'
services:
   db:
     image: mysql:5.7
     volumes:
     - db_data:/var/lib/mysql
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
```

```
        MYSQL_DATABASE: wordpress
        MYSQL_USER: wordpress
        MYSQL_PASSWORD: wordpress

    wordpress:
      depends_on:
      - db
      image: wordpress:latest
      ports:
      - "8000:80"
      environment:
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: wordpress
        WORDPRESS_DB_PASSWORD: wordpress
volumes:
  db_data:
```

Visualizer

- Compose passamos uma label
- logo precisamos definir ela em nosso node: docker node update --label-add dc=UK pop-os

```
version: "3.7"
services:
  web:
    image: nginx
    deploy:
      placement:
        constraints:
        - node.labels.dc == UK
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "8080:80"
    networks:
    - webserver

  visualizer:
    image: dockersamples/visualizer:stable
    ports:
    - "8888:80"
    volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
```

```
    - webserver

networks:
  webserver:
```

 Mini Aplicação