

DOCKER

P/ DEV'S

O que é Docker?

- O docker é um software que reduz a complexidade do setup de aplicações
- Once configuramos containers, que são servidores para rodar nossas aplicações
- Com facilidade, criamos ambientes independentes e que funcionam em qualquer OS
- Trás performance aos projetos

Ex: Ao invés de instalar o php, apache e mysql em um servidor, eu só rode os containers com essas tecnologias e rode a aplicação em cima.

Por que Docker?

- O ambiente de dev e de produção são os mesmos
- A infra se torna código
- Mais performática que VM's

- Instalando o Docker no Ubuntu

- Para instalar o docker no ubuntu, não tem script algum, basta seguir a documentação de instalação.
- Para testar se está funcionando, rode:
 - `# docker ps`
 - `# docker -v`
 - `# docker run docker/library:centos frase aleatoria`

- Containers

- São pacotes de código que podem executar uma ação, como por exemplo rodar uma aplicação PHP.
- Os projetos serão executados dentro dos containers
- Todo container roda em cima de uma imagem.
- múltiplos containers podem rodar juntos, como um para php e outro pra mysql.

- Imagens (Dockerfiles)

- imagem é o "projeto" que será executado pelo container, todas as instruções estarão declaradas nela
- Container é o docker rodando uma imagem.

- Onde encontrar imagens

- imagens podem ser encontradas no repositório do docker
 - hub.docker.com

- Rodando uma imagem

`# docker run nome_imagem`

* ao rodar uma imagem pela 1ª vez, ele verifica se a mesma não está em cache, se não estiver, baixa

D	S	T	Q	Q	S	S
---	---	---	---	---	---	---

↳ flag para que o container continue rodando

15/10/21

Ex: # docker run -it ubuntu

↳ flag para interação

- Verificando containers executados

- docker ps → mostra os containers rodando no momento
 - paramos a flag -a, sabemos tudo o que já rodamos

- Docker run -it php

- a flag -it remete a interação, e permite que nos acessemos tal container, como um ubuntu, php, etc.

- Docker run -d nginx

- a flag -d faz com que um container rode em background, nem usar o terminal

- Docker stop nome_container

- para de rodar um container

- Expondo portas com Docker

- Containers docker não tem a capacidade de porta direta
- Para isso precisamos expor portas, com a flag -p
 - Ex: -p 80:80; → porta do apache
 - Dessa maneira, o container estará acessível na porta 80
 - podemos testar com nginx.

- Reiniciando um container

- Quando eu quiser reiniciar um container que sóvies de criar um novo, basta rodar:

docker start <id>

- Docker run --name nome-app nome
 - Esse comando dá um nome a um container
- Docker logs <id>
 - mostra os logs de um container
- Docker rm <id>
 - remove um container da máquina
- log -/ul>
 - apaga um container mesmo que rodando
- O que são imagens?
 - Originadas de arquivos que programamos para que o docker crie uma estrutura que execute determinadas ações em containers
 - Contêm informações como imagem base, diretórios base, porta da aplicação etc.
- Carregar uma imagem
 - Para carregar uma imagem, basta carregar um arquivo Dockerfile em uma pasta que ficará o projeto.
 - pull : imagem base
 - workdir : diretórios da aplicação
 - expose : porta da aplicação
 - copy : carregar arquivos precarregados

D S T Q Q S S

Ex: Criando uma imagem pra node

FROM node → diz que vai usar o node como padrão (base)

WORKDIR /app

COPY package*.json

RUN npm install → roda esse comando

COPY . → tudo

EXPOSE 3000 → expõe essa porta

Cmd ["node", "app.js"] → roda esses comandos

- Executando uma imagem

- O comando é docker build < diretório da img >
- Depois, utilizar o docker run < imagem > para executá-la

* é necessário rodar os outros comandos e flags ou dar run na nossa própria imagem.

* sempre que alterarmos o dockerfile, é necessário buildar de novo

- Docker pull < imagem >

- faz o download de uma imagem

→ versão da imagem (versão menor)

- Docker tag < id > nome-imagem : tag

- dá nome a uma imagem, que será exibida em:

≠ docker image ls

- Docker build -t nome : tag

- renomeia a img ao criar-la

13/11/21

→ tag

D S T Q Q S S

- Docker rmi : -f

- remove a imagem, acerta -f

→ pode ser para
em qualquer cmd.

- Docker system prune --help

- remove tudo o que não está sendo usado

- Docker run -d -p 80:80 ... --rm <name>

- remove automaticamente após dar stop

- Docker cp <container>:/dir ./dir

- pode ser utilizado para copiar um arquivo de um container para um container ou vice versa.

- Docker top <container>

- mostra os processos dentro do container

- Docker inspect <container>

- mostra como um container foi criado

- Docker stats

- mostra o gasto de recursos do docker

- Docker login

- mostra autentica no docker hub, permitindo que subamos imagens

- Docker logout

- o nome já dig.

- Declarar push <imagem>

- Sube uma imagem no hub (é necessário estar logado e ter um repositório criado)

- Volumes

- Uma forma prática de persistir dados em aplicações e não depender de containers para isso
- Toda classe criada por um container é salva nela, quando o container é removido, perdemos os dados.

- Tipos de volumes

- Anônimos : criados pela flag -v, seriam como um nome abstrato
- Nomados : São volumes com nome
- Bind Mounts : Uma forma de salvar dados na máquina, sem o gerenc. do Docker, informamos um diretório para isso.

- Volumes anônimos

- Podemos criar um anonymous volume da seguinte maneira
+ docker run -v /data

↳ diretório que contém o vol. anônimo

- Este container será associado ao vol. anônimo.

- Docker volume ls

↳ mostra todos os volumes

- Volumes nomeados

- Podemos criar um named volume da seguinte maneira:
+ docker run -v nomevol:/data
↳ ex: /var/www/html/messages
- Pode ser visto com:
+ docker volume ls

- Bind mounts

- Também é um volume, porém ele fica em um diretório que nós especificamos
- Conta não criamos um volume, e sim apontamos um dir.
- O comando é:
+ docker run -v /dir:/data:/data
- Assim, se dir na nova máquina terá o volume.
- Podemos utilizar esta técnica para atualizações em tempo real do projeto
- Sem ter que refazer o build a cada atualiz.

- docker volume create <name>

- Cria um volume manualmente, já nomeado

- docker ^{volume} inspect nome

- o nome já fala

- docker volume rm <name>

- o nome já fala

- docker volume prune

- o nome já fala

Networks

- Uma forma de gerenciar a conexão da Docker com outras plataformas ou até mesmo entre containers.
- As redes são criadas separadas dos containers, como os nsl.
- Uma rede torna sempre a comun. entre containers.

- Tipos de conexões

- Externa: conexão com uma API de um provedor remoto.
- Com o host: Conexão com a máquina que está exec. o docker.
- Entre cent.: comunicação que utiliza o driver bridge e permite a comun. entre dois ou mais cent.

- Tipos de rede (drivers)

- Bridge: é o mais comum e default da Docker, utilizando que os containers precisam se conectar.
- Host: Permite a conexão entre um cent. e a máquina que está rodando o docker.
- Macvlan: via MacAddress.
- None: Remove todas as cen. de um cent.
- Plugins: Permite ext. de terceiros para criar outras rede.

docker network ls

- mostra todas as redes da rede sistema

* O docker já vem com 3 redes instaladas necessárias para o funcionamento.

20/11/21

D S T Q Q S S

docker network create <name>

- O nome já fala
- Por padrão é do tipo bridge.

docker network rm <name>

- O nome já fala

docker network prune

- O nome já fala

- Conexão externa

- Os containers podem se conectar livremente ao mundo ext.
- Um caso seria: uma API de código aberto.
- Pode-se acessá-la livremente e utilizar suas classes.

- Conexão com o host

- Pode-se também conectar um container com o host do Docker.
- Como ip do host utilizamos: host, docker, internal

- Conexão entre containers

- Duas imagens distintas rodando em containers separados que precisam se conectar para intercambiar dados no banco por exemplo.
- Vai precisar de uma rede bridge.

docker run --network <name>

- Cria um container com uma conexão previamente criada.

- Conectar container

- Precisamos conectar um container a uma rede (sem ser em sua criação)
- docker network connect <rede> <container>

- Desconectar container

- docker network disconnect <rede> <cont>

- Docker network inspect <name>

- O nome já fala

- Docker Compose

* é necessário instalar o composer no linux

- O Docker Compose é uma ferramenta para rodar múltiplos containers;
- Temos apenas um arquivo de configuração, que organiza tudo pra nós.
- É uma forma de rodar múltiplos builds e runs com um comando

- Criando nosso primeiro Compose

- Primeiro, criamos um docker-compose.yml na raiz do projeto.
- Esse arquivo vai coordenar os cont. e imagens, e permitir algumas chaves muito utilizadas.
 - Version.
 - Services.
 - Volumes.

- Reclame o Compre.

- checker compre up \rightarrow reclame o Compre, exec as inst Lavorita - d de arquivos

- no Docker - Compre .yml

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db_data:/var/lib/mysql

restart: always

environment:

mysql_root_password: wordpres

mysql_database: wordpres

mysql_user: matheus

mysql_password: secret

wordpress:

depends_on:

- db

image: wordpress: latest

ports: 8000:80

restart: always

environment:

wordpress_db_host: db:3306

wordpress_db_user: matheus

wordpress db_password: secret
wordpress db_name: wordpress

volumes:

db_data: { }

docker-compose down

- dá step em todos os containers

- Variáveis de ambiente

- Para isso, vamos definir um arquivo base em env-file.
- As variáveis podem ser chamadas através da sintaxe \${var}
- Util quando queremos emitir dados sensíveis

ex: No arquivo docker sempre, no lugar das env:

env-file:

- ./config/db.env

↳ dizer que o arquivo de variáveis de ambiente.

- Redes no compose

- O compose cria uma rede básica bridge entre os cont.
- Podemos isolar as redes com a chave networks
- Assim, podemos conectar apenas os cont que optarmos.
- Podemos definir drivers diferentes.

* Nós podemos rodar no container, contudo, para rodar nas nossas próprias imagens (dockerfiles), precisamos fazer o build antes do up.

- Build no container

- Podemos gerar o build durante o processo

No docker-compose, no lugar da image:

build: ./dir -> dockerfile

- Bind mount no container

- O volume de bind mount garante atualizações em tempo real das alterações dos containers.
- Basta configurar isso no compose

15/12/21

- Docker Swarm

- Ferramenta para orquestração de containers em diversas máquinas, para projetos de médio e grande porte.

- Orquestr. de containers

- Orquestração é o ato de conseguir gerenciar e escalar os containers da nossa aplicação
- Temos um serviço que rege sobre outros serviços, verificando se os mesmos estão func. como deviam

→ aumentar quando necessário

- Ex: de orquestradores:
- Docker Swarm
 - Kubernetes
 - Apache mesos

- O que é o docker Swarm

- Ferramenta para orquestrar containers
- Permite escalar horizontalmente o projeto
- O famoso cluster

↳ nome dado ao conj. de máquinas paralelas
reclanhas de forma sincronizada.

- O docker já vem instalado, porém desabilitado.

- Conceitos fundamentais:

- Node: é uma instância (máquina) que participa do swarm
- Manager node: node que gerencia os demais nodes
- Worker node: nodes que trabalham em funç. do manager
- Service: conjunto de tasks que o manager manda o worker executar.
- Task: demandas que são executadas nos nodes

- Maneira de executar o Swarm.

- Para usar o swarm, é necessário mais nodes, seu reja, máquinas, para isso, existem duas opções:
 - Aus: Criar conta e rodar alguns servers (grátis e difícil)
 - Docker lab: gratuito e roda no browser, mas expira em 4 horas.

* farei um curso de aus posteriormente *

- Setup do docker lab. com swarm
 - Acessar labs docker
 - Criar vários nodes
 - \$ docker swarm init --advertise-addr <ip>

- Iniciar o swarm
 - \$ docker swarm init --advertise-addr <ip> → novo node
 - node para quem que a inst. virá um node. node sempre
 - também transforma o node em manager.

- Instando node exteror
 - \$ docker node ls.

- \$ docker swarm leave -f → apenas em manager
 - Desfaz o node

- Adicionando novos nodes → gerado ao dar init
 - \$ docker swarm join --token <token> <ip>:<porta>
 - Dessa forma, duas máquinas estarão conectadas.
 - Esta máquina entrará como worker.
 - Toda as aplicações utilizadas no manager serão replicadas em nodes adicionados com o token

- Subindo um novo serviço → container
 - \$ docker service create --name <nome> <imagem>
 - Dessa forma, adicionamos um container novo ao nosso manager
 - Será gerenciado pelo swarm

docker service ls -a no se manejar puede rodar

- lista los servicios

docker service rm <name>*

- Remove un servicio

- Aumentando o numero de replicas

- Podemos crear un servicio con un numero mayor de replicas con # docker service create --name <name>

• Assim, uma task rodará --replicas <numero> <imagen>

→ Inicia replicando ese servicio

→ sigue mas workers

* aula 134 prova que está funcionando*

docker service ps -a -tken manager

- Devuelve o token por terminal

docker node rm <id> (acota -)

- O node sale del servicio

- Utilidades

docker service inspect

docker service ps

docker info

- Redefinindo com swarm

docker stack deploy -c <arquivo.yml> <nome>

- Teremos então o arquivo com o comando executado.

- Aumentando réplicas do stack

- Redefinir criar réplicas nos worker nodes.

docker service scale <nome> = <replicar>

- Fazer service não receber mais tasks

docker node update --availability drain <id>

- O status drain é o que não recebe tasks

• Para que ele volte a obedecer o manager, basta parar o status active

- Atualizar configurações dos nodes ativos

docker service update --image <imagem> <service>

- Criando rede para swarm

- A diferença entre diferentes inst. usa um driver diferente, o overlay;

docker network create

- Depois, ao criar um service, adiciona a flag --network <rede> para incluir as inst. na nova rede.

- Conectar service a uma rede.

docker service update --network <rede> <nome>

- Kubernetes

- Ferramenta de gerenciamento de containers.
- Permite a criação de múltiplos containers em diferentes níveis.
- Escala projetos, formando um cluster.
- Gerencia serviços, garantindo que as aplicações sejam executadas.
- Criado pelo google.

- Conceitos fundamentais

- **Centro de gerenciamento**: onde é gerenciada e controlada das processos dos níveis.
- **Nodes**: máquinas gerenciadas pelo centro de gerenciamento.
- **Deployment**: A transformação de uma imagem/projeto em um pod.
- **Pod**: Um ou mais containers que estão em um nó → um servidor.
- **Services**: serviços que expõem os pods ao mundo externo.
- **Kubectl**: cli para Kubernetes.

- Dependências necessárias

- Kubectl: maneira de executar o K8 = Kubernetes
- Minikube: espécie de simulador de Kubernetes, para que não precisemos de vários computadores ou servidores.

- Kubernetes no Linux

- instalar K8 (dvc) * testar com ≠ kubectl version
- instalar minikube (dvc) ≠ minikube version

- Iniciando o minikube

- ≠ minikube start --driver=<driver>
- ≠ minikube status ↳ check
- para o minikube

- Acessando a dashboard do Kubernetes

- O minikube nos disponibiliza uma dashboard onde podemos ver todos e detalhamento do nosso projeto, como, services, pods entre outros.

minikube dashboard --url.

- Deployment na teoria

- Com ele criamos nosso serviço que vai executar nos pods.
- Definimos uma imagem e um nome, para posteriormente ser replicado entre os servidores.
- A partir da criação do deployment teremos containers replicados.
- Vamos precisar de uma imagem no Hub do docker para gerar um deployment.

- Criar projeto

- Primeiramente, vamos criar um pequeno proj. em flask.
- Buildar a imagem da mesma
- Enviar a imagem para o docker hub. → docker login
- E testar executar em um container.

- Criando nosso deployment.

kubectl create deployment <name> --image=<imagem>

- Checando deployments.

- # kubectl get deployments!
- # kubectl describe deployments.

- Checando Pods

- Os pods são componentes muito importantes onde os containers realmente são executados.
 - ≠ kubectl get pods
 - ≠ kubectl describe pods

- Confi. de K8s

- Podemos configurar como o K8S está configurado
 - ≠ kubectl config view.
- No menu com names receberemos import. barra das no mini-kube, que é por onde o kubernetes está sendo executado.

- Services Teoria

- As aplicações do kubernetes não tem conexão com o mundo exterior.
- Por isso precisamos criar um Service, que é o que possibilita expor os pods.
 - Isso acontece pois os pods são criados para serem destruídos e perderem tudo, ou seja, os dados gerados nelas também são apagados!
 - Então o service é uma entidade separada dos pods, que expõe elas a uma rede.

- Criando service

- Para criar um service é preciso mapear os pods disponíveis utilizando o comando:

kubectl expose deployment < nome >

-- type=< tipo >

-- port=< porta >

- Recuperaremos o nome do deployment já criado.

- O tipo de service, há várias para utilizar, porém o Load Balancer é o mais comum, onde todos os pods são expostos.

- É uma porta para o serviço ser consumido

- Gerando ip de acesso

- Podemos acessar o novo service com o comando:

minikube service < nome >

- Desta forma o IP aparece no novo terminal

- Pronto, praj reclamado.

- Verificando meus serviços

kubectl get services

kubectl describe services/< nome >

- Replicando a aplicação

- Vamos aprender a como utilizar outros pods, replicando assim a nova aplicação

kubectl scale deployment/< nome > --replicas=< numero >

kubectl get pods → checar

Serve para aumentar seu dim.

Basta alterar o num. de replicas

20/12/21

D S T Q Q S S

- modo declarativo

- O que estaremos fazendo só agora, se chama modo imperativo, que é o modo que precisamos digitar todos os comandos para a pane.
- Pode também se modo declarativo, que semelhante ao docker sempre, é mandado um yaml com todas as instruções ao k8s, que pode ser exec. com o comando.

- Chaves mais utilizadas

- apiVersion: versão utilizada da ferramenta.
- kind: tipo do arquivo (Deployment, Source);
- Metadata: descrever algum objeto, envolvendo chaves como a chave name.
- Replicas: numero de replicas de nodes/pods.
- Containers: definir as especificações de containers como nome e imagem.

- Criando nosso arquivo.

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-app-deployment

spec:

replicas: 4

selector:

matchLabels:

app: flask-app

template: ~~~~~

template:

metadata:

labels:

app: flask-app

spec:

containers:

- name: flask

image: reneholz/flask:1.0

* encerrando por aqui para não achar ruim

D	S	T	Q	Q	S	S
---	---	---	---	---	---	---

20/12/21

kubectl get rs

- Trás o numero de replicas

- Atualização de imagem

- Para atualizar a imagem vamos precisar da nome da versão que é dada no Dashboard dentro do hub.

- A imagem também deve ser diferente da versão atual.

- Vamos precisar subir uma nova Tag no hub.

- Depois podemos:

kubectl set image deployment/<name>

<name-cont> = <nova-img>

- Deployer alteração

- O comando para verificar uma alteração é:

kubectl rollout status deployment/<name>

- Com ele e se # kubectl get pods # podemos encontrar o problema.

- Para voltar a alt. usamos:

kubectl rollout undo deployment/<name>

- Deletar um service

kubectl delete service <name>

- Dessa maneira novos pods não terão mais cen. externa

- Da reja, não poderemos mais acionar eles.

Kubectl delete deployment <name>

- Deleta os deployment.

- Subnets

- Uma sub-rede de uma rede, de forma geral, inclui todos os computadores em uma localidade específica.
- Uma subnet é criada para cada grupo de dispositivos.
- Existem 2 tipos de subnets: pública → tem acesso à internet private → n/ tem acesso à internet
- Seu tipo é definido de acordo com a tabela de reteas