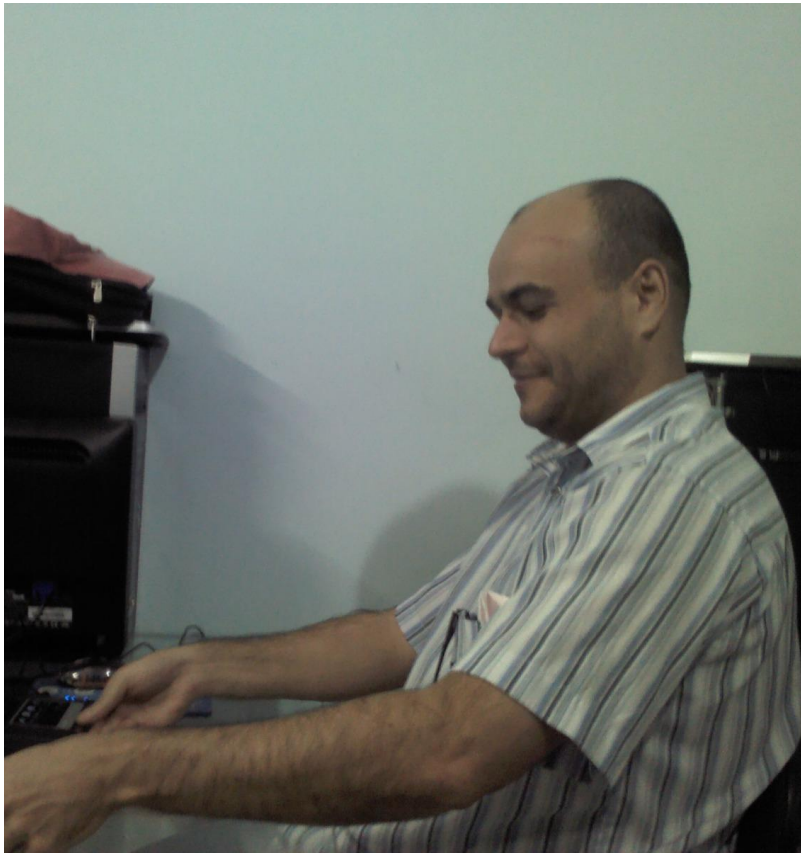
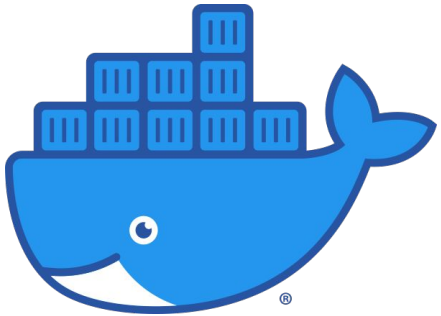
MEGANEWS®

OPORTUNIZANDO TALENTOS


SISTEMAS
CERTIFICADO
Delphi




SISTEMAS
Uso Consciente de Energia
Sukita



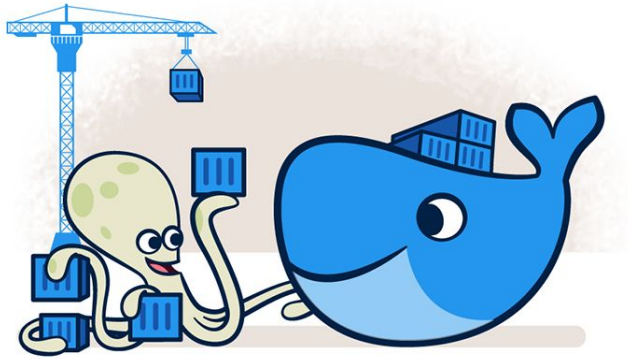
Introdução ao Docker

Por Julio Novaes e Guilherme Gomes

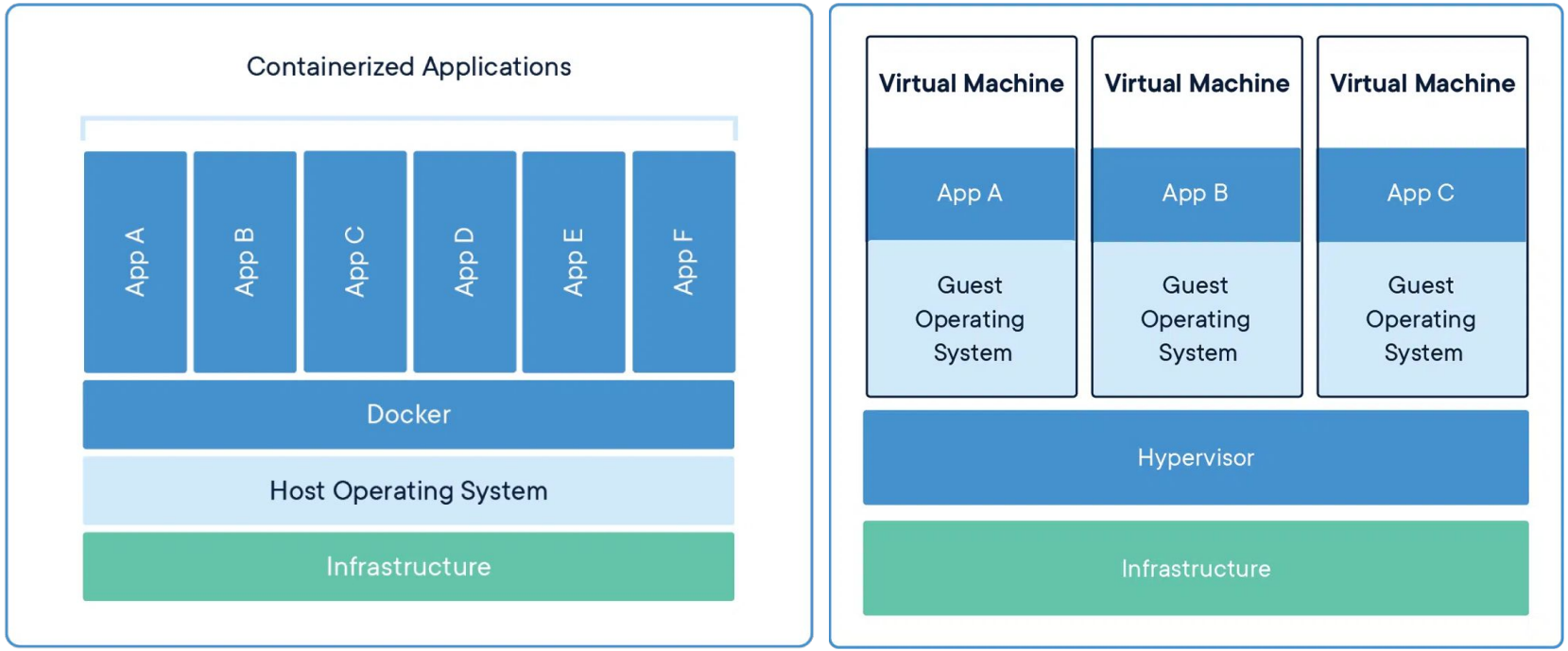
O que é o Docker?

- Plataforma de código aberto que permite o **empacotamento** e execução de aplicativos de maneira fácil e consistente em diferentes ambientes de computação.

- Utiliza virtualização de contêiner.

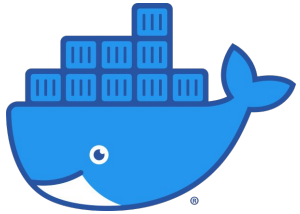


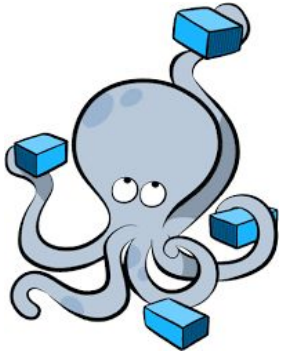
Contêiner vs VM



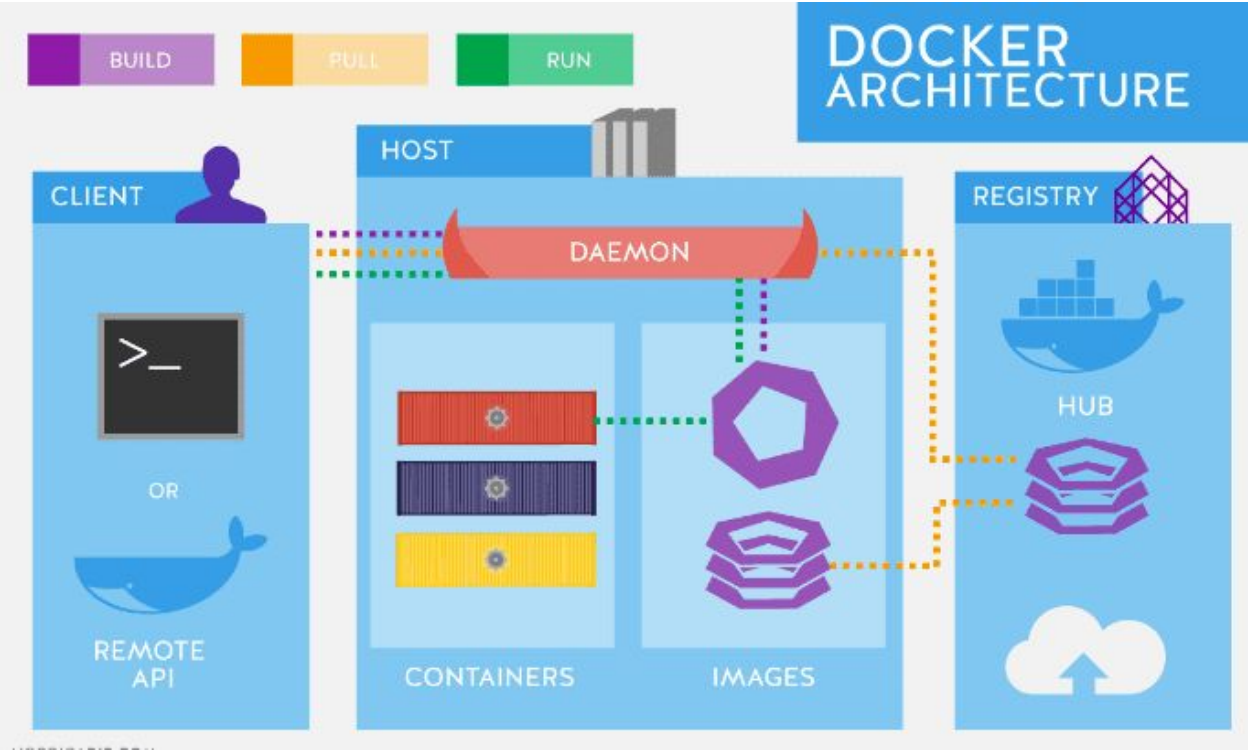
Aplicações para o Docker

- Implantação de Aplicativos
- Ambientes de desenvolvedor
- CI/CD
- Escalonamento de aplicativos
- Microsserviços
- Ambientes de teste
- Kubernetes



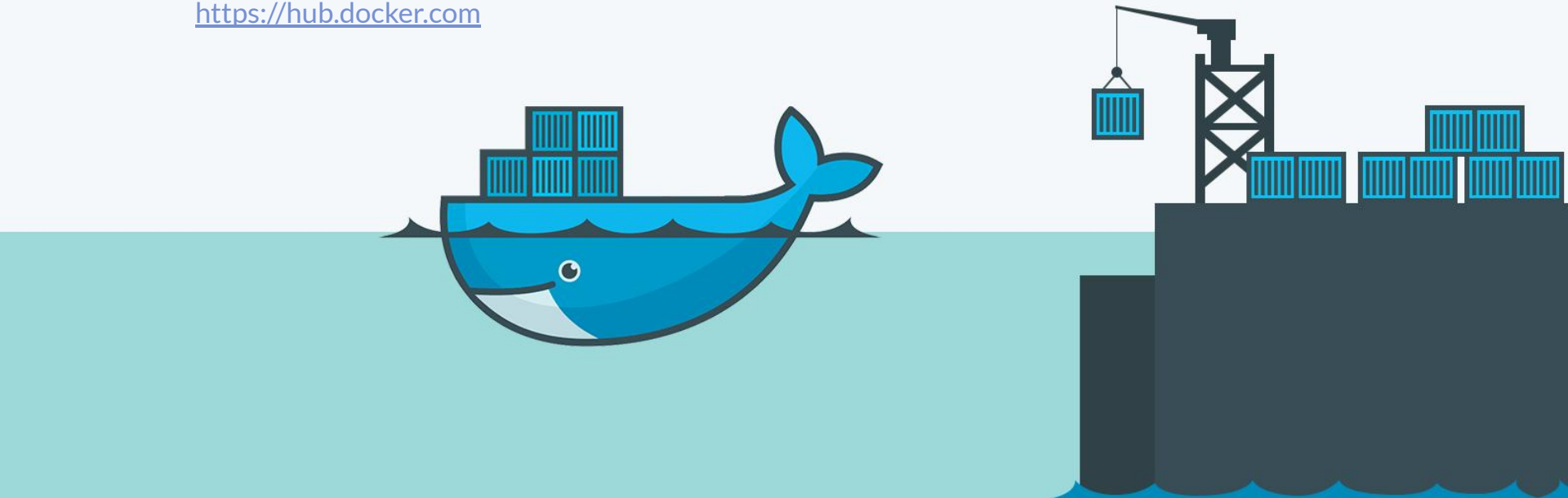
Arquitetura do Docker



Docker Hub

O Docker Hub é um serviço online oferecido pela Docker que permite o armazenamento e compartilhamento de imagens de containers Docker.

https://hub.docker.com



Playground Docker

https://labs.play-with-docker.com

Instalação

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

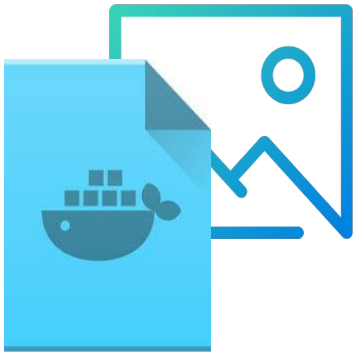
Executando seu Primeiro Container

```
1 docker pull ubuntu:22.04
2 docker run -it ubuntu:22.04
```

Pull baixa uma imagem e **Run** a executa.

Executar `docker run` sem executar um pull antes, baixa a imagem automaticamente.

Imagens de Container



O que são Imagens de Container

Imagens de contêiner são um tipo de imagem virtual que contém um sistema operacional e os aplicativos e arquivos necessários para executar um determinado software.

Elas são criadas a partir de um Dockerfile.

Dockerfile

É um arquivo de texto que dá instruções ao Docker para criar uma imagem.

Ele parte de uma imagem base que geralmente é um Sistema Operacional e executa diversas configurações e comandos para configurá-la para executar uma dada aplicação.

A compilação de uma imagem ocorre em camadas, onde cada instrução do arquivo representa uma camada.

Prática - Seu Primeiro Dockerfile

Crie um arquivo chamado “Dockerfile”.

```
1  FROM ubuntu:22.04
2
3  RUN apt update && apt install sl -y
4
5  ENTRYPOINT ["/usr/games/sl"]
```

Execute:

```
[~]► docker build -t minha-primeira-imagem .
```

Verifique se a imagem foi compilada com o comando “docker images”:

```
[guilherme.gomes][~]► docker images
REPOSITORY              TAG                 IMAGE ID       CREATED          SIZE
minha-primeira-imagem   latest              779fb9a2e93b   2 minutes ago    119MB
ubuntu                  22.04               99284ca6cea0   3 weeks ago      77.8MB
```

Execute a imagem compilada com:

```
[~]► docker run -t minha-primeira-imagem
```

Instruções Dockerfile

FROM <imagem>:<tag> - Define qual será a imagem base que executará as próximas instruções.

WORKDIR <diretório> - Define o diretório de trabalho atual.

COPY <origem> <destino> - Copia arquivos do host para a imagem.

RUN <comando> - Executa um determinado comando.

ENTRYPOINT <comando> - Define o comando que será executado na execução de um “docker run” (só deve haver um por arquivo)

EXPOSE <porta> - Documenta ao usuário qual porta do container deve ser encaminhada.

Exercício - Implantando uma Aplicação

Copie o código da aplicação no repositório do workshop, e salve-o em uma pasta separada na sua workstation como “main.go”.

Com as informações abaixo, crie um Dockerfile para essa aplicação, de forma que seja possível fazer requisições a partir do host.

Imagem base para o Go: golang:1.20-alpine

Comando de build do Go: go build -o <nome-do-executavel> main.go

Após criar o Dockerfile compile e execute a imagem para testar.

Resolução

```dockerfile
FROM golang:1.20-alpine AS build

WORKDIR /app

COPY . .

RUN go build -o server main.go

FROM alpine:3.17

WORKDIR /app

COPY --from=build /app/server server

ENTRYPOINT ["/app/server"]
```

Um Dockerfile que utiliza mais de uma imagem base, é chamado de “multi-stage”, ou multi-estágio, onde cada imagem base representa um estágio.

Prática - Docker Hub

Para enviar suas imagens ao Docker Hub:

1. Faça login no seu Docker CLI:

```
[guilherme.gomes][~]► docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID
, head over to https://hub.docker.com to create one.
Username: gomessguii
Password:
WARNING! Your password will be stored unencrypted in /home/guilherme.gomes/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Para enviar suas imagens ao Docker Hub:

2. Compile sua imagem com seu nome de usuário do Docker Hub.

```
[~]► docker build -t gomessguii/minha-primeira-imagem .
```

3. Envie a imagem

```
[~]► docker push gomessguii/minha-primeira-imagem
```

4. Verifique as imagens enviadas em https://hub.docker.com/seu-usuario

Tags de Versão da Imagem

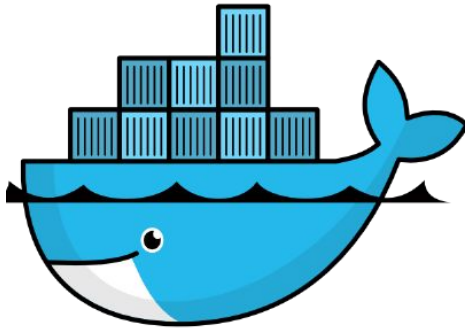
Ao compilar uma imagem é possível colocar uma tag de versão em seu nome, exemplo:

docker build -t minha-imagem:minha-tag .

Essa tag é utilizada para diferenciar a versões de uma mesma imagem. Como golang:1.19 e golang:1.20. Mas também pode ser utilizada para diferenciar a distro que aquela imagem usa, por exemplo. Por isso golang:1.20 é diferente de golang:1.20-alpine.

Quando não é informada uma tag aos comandos, o docker busca pela tag “latest” por padrão.

Containers do Docker



O que é um container na prática?

Na prática, um container é uma imagem baixada e preparada para execução. Uma única imagem pode gerar inúmeros contêineres, em inúmeros hosts.

Quando uma aplicação é implantada em produção com Docker, ela se torna um (ou vários) contêineres. Por isso, é crucial aprendermos a lidar com eles.

Comandos de Containers

docker ps ou **docker container ls** - Lista todos os contêineres em execução no momento. Pode ser utilizada a flag -a para listar os contêineres parados também.

docker stop e **docker start** - Seguidos do nome ou ID do contêiner, esses comandos param e iniciam um container, respectivamente.

docker rm <nome_ou_id_do_container> - Remove um contêiner parado.

Comandos de Containers

docker exec <nome_ou_id_do_container> <comando> - Executa um comando dentro de um container

docker run -it - Executa um container de forma interativa

docker run -d - Executa em segundo plano

docker logs <nome_ou_id_do_container> - Busca os logs de um container

Debugando um Container

Quando encontramos algum problema em aplicações Docker, pode ser necessário debugá-las. Para isso existem algumas coisas que podemos verificar.

Encaminhamento de Portas

Um problema muito comum com aplicações Docker é um encaminhamento de portas mal configurado. O que leva a aplicação a ficar inacessível externamente.

Para verificar o encaminhamento, utilizamos “docker ps”, veja:

```
[guilherme.gomes][ex02](main)► docker ps
CONTAINER ID   IMAGE                      COMMAND          CREATED          STATUS          PORTS                                      NAMES
47161195fb8d   minha-primeira-aplicacao   "/app/server"    2 seconds ago    Up 1 second     8000/tcp                                   encaminhamento-desconfigurado
d5e50a8dbb0c   minha-primeira-aplicacao   "/app/server"    31 seconds ago   Up 30 seconds   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp  encaminhamento-configurado
```

Logs do Container

Se houver algo de errado com a aplicação em si, ela deve lhe dizer pelos logs. Portanto, é importante saber como checá-los numa aplicação docker.

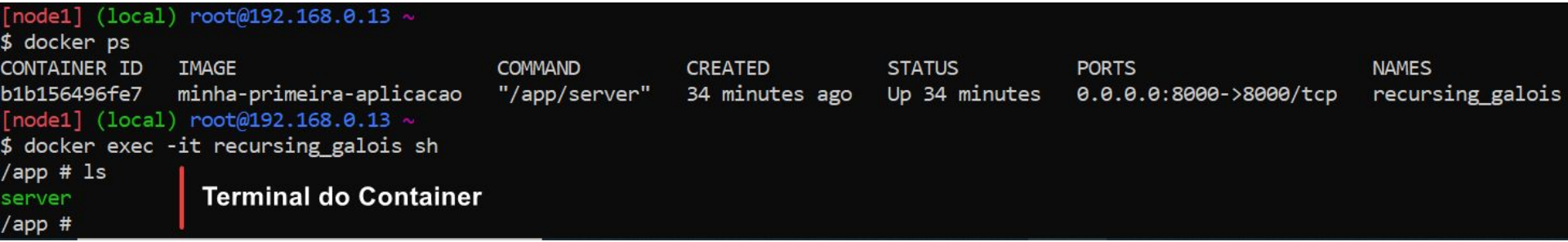
Para checar os logs, utilizamos o comando “docker logs <nome_ou_id_do_container>”, veja:

```
[node1] (local) root@192.168.0.13 ~
$ docker ps
CONTAINER ID   IMAGE                      COMMAND          CREATED          STATUS          PORTS                    NAMES
b1b156496fe7   minha-primeira-aplicacao   "/app/server"    29 minutes ago   Up 29 minutes   0.0.0.0:8000->8000/tcp   recursing_galois
[node1] (local) root@192.168.0.13 ~
$ docker logs rec
recursing_galois   recursing_mclaren
[node1] (local) root@192.168.0.13 ~
$ docker logs recursing_galois
listening on :8000
```

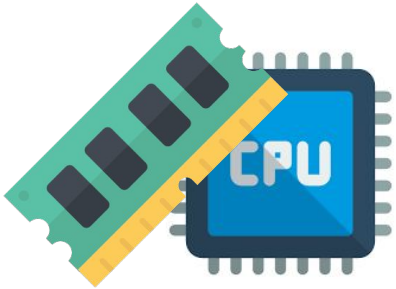
Terminal do Container

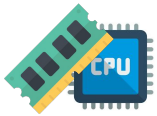
Outro erro muito comum em aplicações docker é não copiar todos os arquivos necessários para a imagem, ou copiá-los para o diretório errado. Uma forma de verificar isso é: com o container em execução, abrir seu terminal e navegar pelos arquivos.

Para isso utilizamos o comando “docker exec” de forma interativa para abrir o terminal. Veja:



Gerenciamento de Recursos



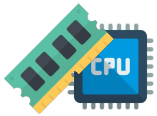


Limitação de Recursos

Ao executar diversas aplicações em diferentes containers num mesmo host, a depender das aplicações, isso pode deixar o host lento. E limitar os recursos de cada container pode ser uma forma de contornar esse problema.

Para isso, existem duas flags que podemos atribuir ao comando “docker run” para limitar o uso de memória e CPU. São eles:

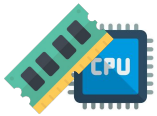
- “cpus” - limita a quantidade de CPUs utilizadas (em unidade de CPUs)
- “memory” - limita a quantidade de memória RAM



Exemplo

```
docker run --cpus=".5" minha-primeira-aplicacao
```

```
docker run --memory="50m" minha-primeira-aplicacao
```



Estatísticas

Uma forma de checar o uso de recursos de cada container é o comando “docker stats”, que mostra em tempo real o uso de CPU, memória e rede dos containers.

Ele tem duas formas:

```
# para checar todos os contêineres de uma só vez
docker stats
# para checar contêires específicos
docker stats container1 container2
```

Exemplo

- docker stats

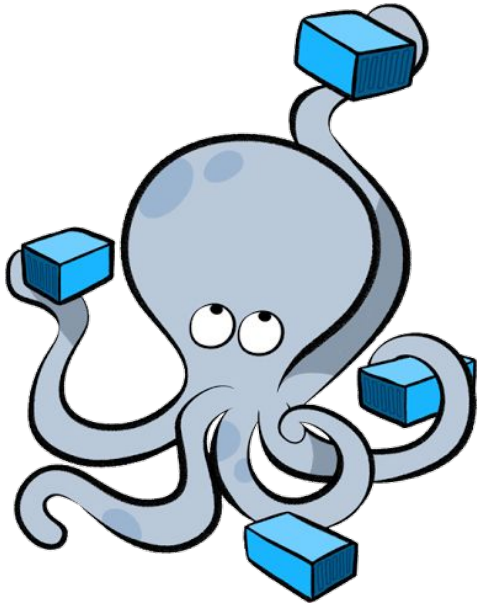
```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O         PIDS
96b22962371f   com-limitacao     0.00%     5.18MiB / 7MiB        74.00%    2.18kB / 0B   36.7MB / 729kB    6
51efb74a5da4   sem-limitacao     0.00%     7.859MiB / 15.36GiB   0.05%     3.11kB / 0B   5.14MB / 0B       6
```

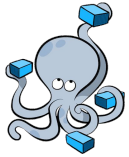
- docker stats com-limitacao

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O       PIDS
96b22962371f   com-limitacao     0.00%     5.176MiB / 7MiB     73.94%    3.3kB / 0B  36.7MB / 729kB  6
```

Retorna estatísticas apenas do container chamado “com-limitacao”.

Docker Compose



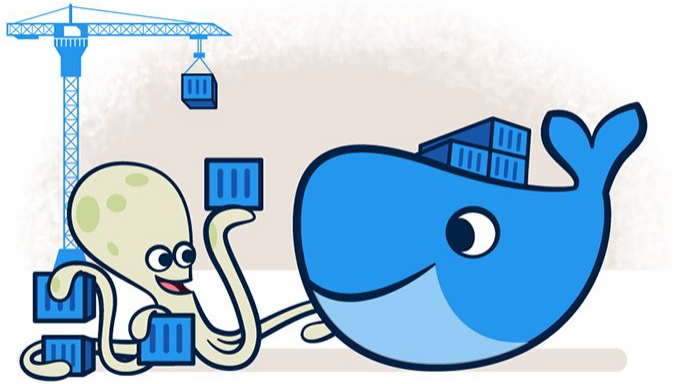


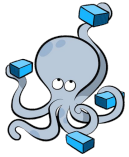
O que é?

O Docker Compose é uma ferramenta que permite definir e gerenciar aplicativos compostos por vários containers Docker. Ele utiliza um arquivo YAML para configuração, onde cada container é denominado serviço.

Em cada serviço pode ser configurado:

- Imagem;
- Encaminhamento de portas;
- Rede;
- Variáveis de ambiente;
- Volumes (persistência de dados);
- Gerenciamento de recursos;
- E mais.



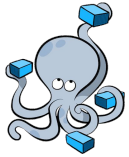


Por que usar o Docker Compose?

O Docker Compose não é realmente necessário, pois tudo que ele faz pode ser feito diretamente com o docker, mas ele facilita e muito a configuração de aplicações multi container, tornando mais fácil de desenvolver e executar.

Exemplo:

```yaml
version: "3"
services:
  servico1:
    imagem: minha-imagem
  servico2:
    imagem: minha-outra-imagem
```



Comandos Docker Compose

- **docker-compose create** - Baixa as imagens a serem utilizadas (quando não presentes) e cria os containers.
- **docker-compose start** - Inicia um serviço já criado.
- **docker-compose stop** - Pára um serviço em execução.
- **docker-compose rm** - Remove os containers parados.
- **docker-compose up** - Cria e inicia os containers e mostra os logs dos serviços de forma interativa. Útil para a primeira vez que subir os serviços, para verificar se sua inicialização ocorre como esperado. (`-d` em segundo plano, `–build` compila o Dockerfile vinculado)

Prática -
Docker Compose

Siga para o repositório do workshop e siga as instruções