Advanced Institute for Artificial Intelligence

Preparação do Ambiente

Advanced Institute for Artificial Intelligence – AI2

https://advancedinstitute.ai

Containers Docker

Links e Referências

- https://docker-curriculum.com/
- https://docs.microsoft.com/pt-br/visualstudio/docker/tutorials/docker-tutorial
- https://emuvm.com/
- https://www.virtualbox.org/
- https://www.vmware.com/br.html

Agenda

- Conceitos e definições
- Arquitetura Docker
- Criação e manipulação de containers
- Construindo imagens personalizadas

Aplicações e servidores

Antigamente, era necessário vários servidores para rodar cada uma das aplicações.

Figure: Serviços e aplicações

Virtualização

As máquinas virtuais (VMs) foram possíveis de ser criadas graças a uma tecnologia chamada ***Hypervisor***, que funciona em cima do sistema operacional, permitindo a virtualização dos recursos físicos do nosso sistema.

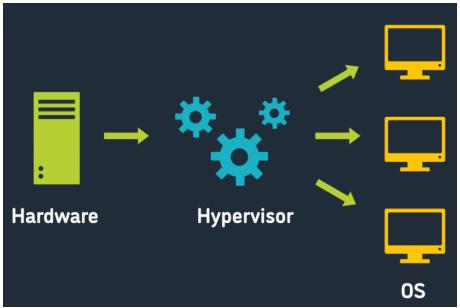

Figure: Hypervisor

Virtualização

Como temos uma máquina virtual que está acessando uma parte do nosso hardware como um todo, conseguimos colocar dentro dela uma das nossas aplicações. E replicar isso, criando mais máquinas virtuais com outras aplicações:

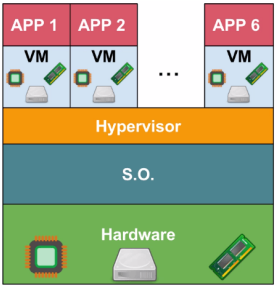

Figure: Máquinas Virtuais

Como nem tudo são flores...

- Em partes, uma VM resolve os problemas do uso de vários servidores físicos;
- Porém, também possuem suas limitações;
- Para podermos hospedar a nossa aplicação em uma máquina virtual, também precisamos instalar um sistema operacional em cada uma delas:

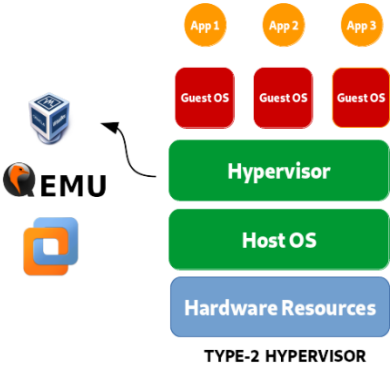

Figure: Máquinas Virtuais

Então, como melhoramos isso???

Então, como melhoramos isso???

Através de Containers

Figure: Contêiner

Containers

Definição

- Mais leve e rápido;
- Sem custos para manter vários S.Os;
- Inclusão e Exclusão praticamente em tempo real.

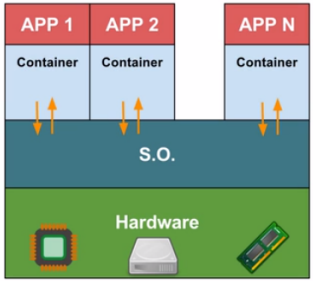

Figure: Aplicações utilizando Containers

Containers

Mas...

- Já parou para pensar porquê eu preciso de Containers?
- Se tudo já funciona dessa forma em nossas máquinas.

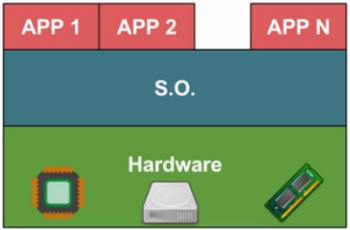

Figure: Aplicações utilizando Containers

É problema puxando problema rs

- Dois ppps utilizarem a mesma porta da rede;
- Um app consumindo todos os recurso da CPU, irá travar os demais serviços;
- Cada app necessita de uma determinada versão para ser executado;
- Uma *Query* de BD congelando o sistema, reinicio o SO?

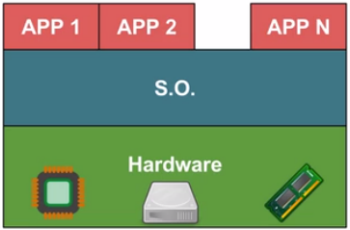

Figure: Aplicações utilizando Containers

Utilizando Containers

- Melhor distribuição de entre os apps;
- Melhor controle de uso para os recursos de CPU, Rede, HDs.. etc;
- Mais agilidade para derrubar e subir um container;
- Simples para manipular diferentes versões e bibliotecas entre eles;
- Muito mais leve que uma VM.

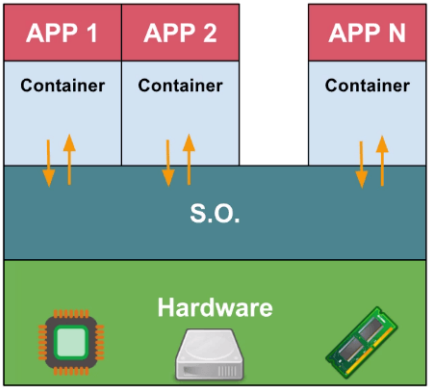

Figure: Aplicações utilizando Containers

Mais Vantagens

- Têm o mesmo desempenho que o código executado no sistema operacional host:
  - Até três vezes mais desempenho do que as máquinas virtuais, quando executados no mesmo hardware).
- Tempo de inicialização em milissegundos (em comparação com minutos para uma máquina virtual).
- Requerem pouca memória RAM
- São definidos usando código
  - Podemos tirar vantagem de sistemas de controle de código como o Git.
- Incentivam o reúso, para que você possa minimizar o retrabalho dispendioso.

Desvantagens

- Sempre são executados no sistema operacional Linux (os containers compartilham o sistema operacional do host):
  - Máquinas virtuais podem executar um sistema operacional diferente para cada máquina virtual
- Os containers usam isolamento no nível do processo
  - potencialmente menos seguro do que as máquinas virtuais totalmente isoladas

Introduzindo Dockers

Agora que já vimos a diferença entre máquinas virtuais e containers, chegou a hora de conhecermos o *Docker*.

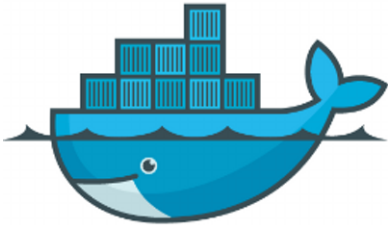

Figure: Docker, a tecnologia de Containers

Docker Engine

Definição simples

- Uma tecnologia de intermédio entre os S.Os e os Containers;

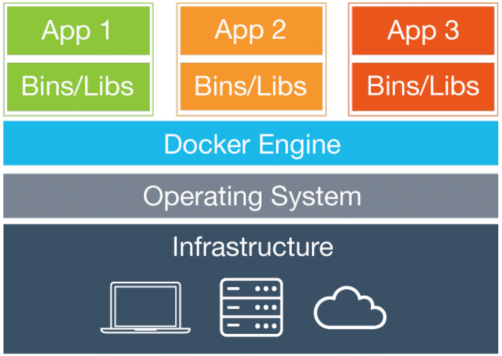

Figure: Aplicações utilizando Containers

Tecnologias de Dockers

- **Docker Compose:** Organiza vários containres

Figure: Docker Compose

Tecnologias de Dockers

- **Docker Swarm:** Coloca múltiplos Dockers Host's para trabahlar em um cluster

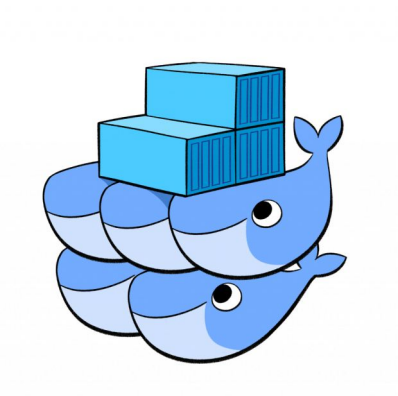

Figure: Docker Swarm

Tecnologias de Dockers

- **Docker Hub:** Repositório com mais de 250k imagens diferentes de containers

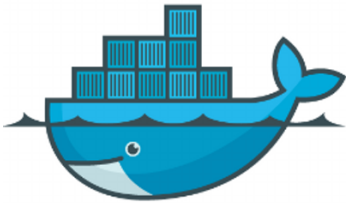

Figure: Docker Hub

Tecnologias de Dockers

- **Docker Machine:** Permite a instalação e configuração de hosts virtuais.

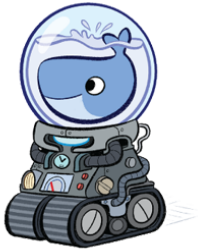

Figure: Docker Machine

VM X Containers

Através de Containers

Figure: Contêiner

vamos colocar as mãos na massa!!!

- Vamos criar uma env do Conda para instalação e configuração dos pacotes necessários;
- Através do comando **conda create n (nome da env)** crie seu ambiente para testes do Docker;
- Ative sua env com o comando **conda activate (nome da env)** .
- Siga os passos para instalação do ambiente Docker no arquivo de instalação compartilhado durante a aula.

Mas o que são imagens?

Conceitos

- **Imagem**: um conjunto estático de arquivos binários que armazenam todas as informações necessárias para iniciar um contêiner
- **Contêiner**: um sistema operacional isolado usando conteinerização (neste caso via Docker) para rodar em um sistema operacional host (neste caso MacOS)

Importante

É importante ressaltar que um contêiner é uma instância em execução de uma imagem.

Imagens

- Materialização de um modelo de um sistema de arquivos
- Produzido através de um processo de build;
- Representada por um ou mais arquivos e pode ser armazenada em um repositório como Github;
- O Docker utiliza file systems especiais para otimizar o uso, transferência e armazenamento das imagens, containers e volumes.
  - O principal é o AUFS, que armazena os dados em camadas sobrepostas, e somente a camada mais recente é gravável.

Arquitetura

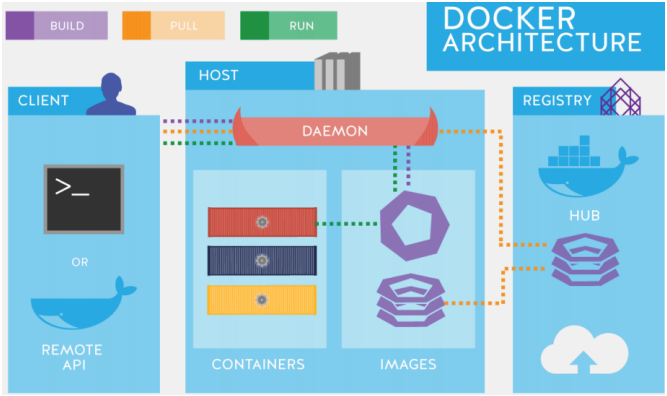

Figure: Arquitetura Docker

Imagens

Quando executamos o comando *docker run hello-world, ele está fazendo o seguinte:*

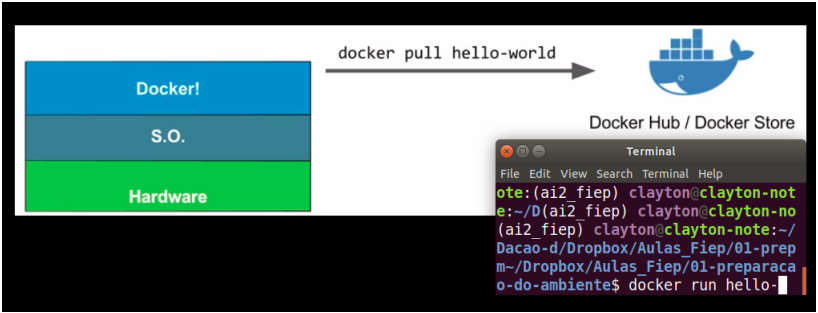

Figure: Contêiner

Parâmetros de Execução

- Execução Interativa:
  - **--interactive** - isso nos permitirá digitar comandos de forma interativa no contêiner
  - **--tty** - aloca um pseudo-TTY que permitirá que o contêiner imprima sua saída na tela.

```
1  > docker run --interactive --tty ubuntu:20.04 bash
2
3  Unable to find image 'ubuntu:20.04' locally
4  20.04: Pulling from library/ubuntu
5  a70d879fa598: Pull complete
6  c4394a92d1f8: Pull complete
7  10e6159c56c0: Pull complete
8  Digest: sha256:3c9c713e0979e9bd6061ed52ac1e9e1f246c9495aa063619d9d695fb803
       9aa1f
9  Status: Downloaded newer image for ubuntu:20.04
10 root@e2dd9abd3559:/#
```

Parâmetros de Execução

```
root@e2dd9abd3559:/# uname -r
4.19.121-linuxkit

root@e2dd9abd3559:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.2 LTS"
```

Principais comandos:

- Listar Imagens:
  - `docker image ls`
- Listar Containers
  - `docker ps -a`
- Remover Containers/Imagens
  - `docker [image] rm 6f0684d58dca`
- Mapeamento de Volumes
  - `docker run -it -v [host]:[container] ubuntu bash`

Imagens Personalizadas

- A imagem é a abstração da infraestrutura em estado somente leitura, de onde será instanciado o contêiner.
- Imagens podem ser oficiais ou não oficiais
- As imagens oficiais são mantidas pela empresa docker e disponibilizadas no *docker hub*;

Importante

- Todo contêiner é iniciado a partir de uma imagem
- Nunca teremos uma imagem em execução
- Um contêiner só pode ser iniciado a partir de uma única imagem

Imagens Oficiais

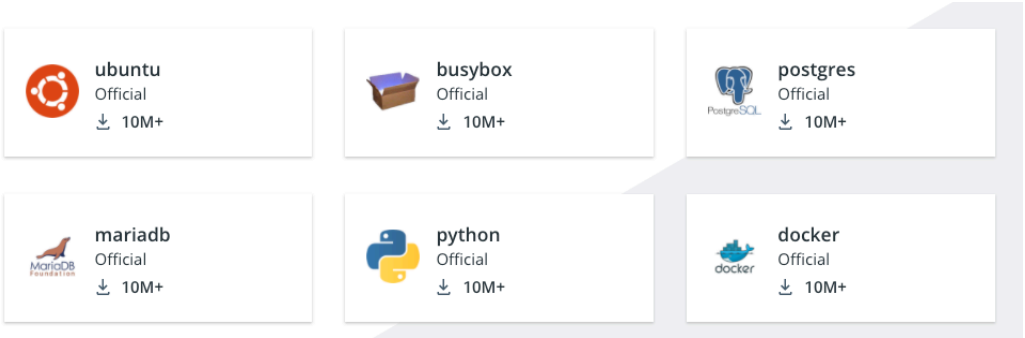

Figure: Imagens oficiais no Docker Hub

Imagens Personalizadas

- O objetivo das imagens oficiais é prover um ambiente básico;
  - um ponto de partida para criação de imagens pelos usuários
- As imagens não oficiais são mantidas pelos usuários que as criaram
- Nomeclatura:
  - Nome de uma imagem é composto por duas partes: *repositório* e *tag*;
    - **`ubuntu:20.04`**: repositório `ubuntu` e tag `20.04`
  - Cada dupla **`repositório:tag`** representa uma imagem diferente;

Utilizando o comando `commit` para criação de imagem:

- Criando um contêiner e realizando alguma modificação na imagem padrão:

```
1 > docker container run -it --name ubuntu-fun ubuntu:20.04 bash
2 root@fb4cdc61c273:/# apt update
3 root@fb4cdc61c273:/# apt install cowsay fortune -y
4 root@fb4cdc61c273:/# exit
```

Utilizando o comando `commit` para criação de imagem:

- Agora vamos criar uma nova imagem utilizando o comando **`commit`**:

```
> docker container commit ubuntu-fun ubuntu:fun
sha256:d88fd6d76bcba0de4c4bb2304330d0b662a5cadd4db447ba83ae54fc6bba848c
> docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
ubuntu       fun       d88fd6d76bcb   4 seconds ago    149MB
```

Utilizando o comando `commit` para criação de imagem:

- Vamos testar a nova imagem:

```
> docker run --rm ubuntu:fun bash -c "/usr/games/fortune | /usr/games/cowsay"
 ----------------------------------------
/ Q: What's tan and black and looks great \
\ on a lawyer? A: A doberman.             /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Dockerfiles:

- Conjunto de instruções aplicadas em uma imagem para geração de outra;
- Controle de diferenças entre uma imagem (base), e a imagem que se deseja criar;

```
1 > cat Dockerfile
2 FROM ubuntu:20.04
3 RUN apt-get update && apt-get install cowsay fortune -y
4 COPY arquivo_teste /tmp/arquivo_teste
5 CMD bash
```

Diretivas do Dockerfile:

- `FROM`: informa qual a imagem base
- `RUN`: informa quais comandos serão executados durante a criação da imagem
- `COPY`: copia arquivos do *host* para a imagem;
- `CMD`: informa qual comando será executado por padrão, caso nenhum seja informado na inicialização do contêiner

Construção da imagem:

```
> docker image build -t ubuntu:fun .
Sending build context to Docker daemon   2.56kB
Step 1/4 : FROM ubuntu:20.04
 ---> 26b77e58432b
Step 2/4 : RUN apt-get update && apt-get install cowsay fortune -y

 ...

Step 3/4 : COPY arquivo_teste /tmp/arquivo_teste
 ---> dea0a06e75d4
Step 4/4 : CMD bash
 ---> Running in 9e38da969f42
Removing intermediate container 9e38da969f42
 ---> 5a1f0278b1c3
Successfully built 5a1f0278b1c3
Successfully tagged ubuntu:fun
```

Outras Diretivas do Dockerfile:

- `ADD`: faz o mesmo que o `COPY`, porém permite que a cópia seja feita de uma URL
- `ENTRYPOINT`: Define um executável (e argumentos padrão) a ser executado quando o contêiner é iniciado - não é redefinido pela linha de comando
- `ENV`: Define variáveis de ambiente dentro da imagem;
- `WORKDIR`: Define o diretório de trabalho para qualquer `RUN`, `CMD`, `ENTRYPOINT`, `ADD` ou `COPY` subsequente;