Advanced Institute for Artificial Intelligence

Cloud ML

Advanced Institute for Artificial Intelligence – AI2

https://advancedinstitute.ai

Cloud ML

Agenda

- Origem e Motivação
- Modelo de Uso
- Estratégias de Uso
- Virtualização
- Docker

Cloud ML

Grid Computing (computação em grade ou computação em malha) [1]

- Iniciativa pioneira para computação distribuída
- Grid é uma analogia a rede elétrica, poder computacional a disposição, no mesmo molde que energia elétrica
- Aplicação prática para projetos científicos, que utilizavam supercomputadores em conjunto

[1] I. Foster, "The anatomy of the grid: enabling scalable virtual organizations," Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001

Cloud ML

Grid Computing (computação em grade ou computação em malha)

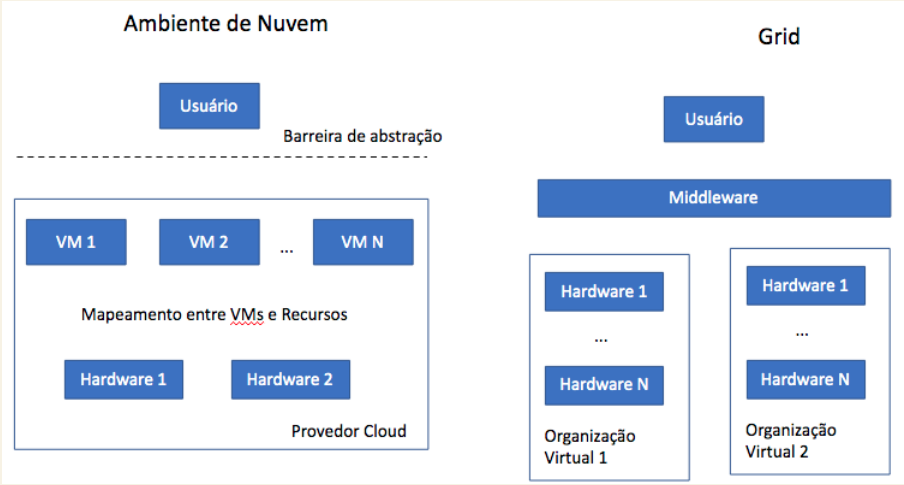
- Esse modelo é baseado no uso do hardware diretamente
- Alta complexidade de uso. Desenvolvedores responsáveis por montar e executar a aplicação
- Limitação para uso de Virtualização:
  - Complexidade de implementação
  - Limitação de hardware
    - Hardware monoprocessado
    - Time sharing entre virtualização e processo, tornaria inviável a execução de aplicações

Cloud ML

Transição de grid para cloud (Motivações):

- Inviabilidade do Uso do grid para aplicações comerciais
- Nós computacionais com múltiplos processadores
  - Possibilidade de executar virtualização de modo eficiente
- Avanços nas tecnologias de virtualização
  - Um nó inteiro poderia ser controlado por um sistema de virtualização

Diferenças entre Grid e Cloud



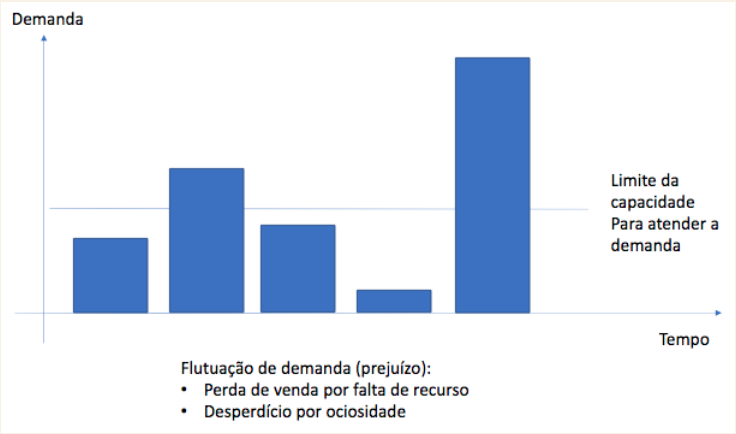
Cloud ML

Transição de grid para cloud (Motivações):

- Possibilidade de explorar o balanço entre o número de máquinas virtuais oferecidas em relação ao desempenho da máquina virtual
- Não precisa de um middleware complexo. Interface simples para interagir com recursos
- Possibilidade clara de otimizar custos com TI de acordo com as demandas por processamento

Cloud ML

Desafio do rateio entre demanda e poder computacional



Cloud ML

Modos de uso de recursos oferecidos por um ambiente de nuvem:

- IaaS
- PasS
- SaaS

Ambientes podem oferecer todos esses modos ou se especializarem em alguns deles

Cloud ML

IaaS (Infrastructure as a Service)

- Ambiente oferece os recursos básicos para montar um ambiente de TI
  - Recursos de rede
  - Computadores (virtuais ou em hardware dedicado)
  - Espaço de armazenamento de dados recursos diversos
- Recursos podem ser criados, destruídos e combinados de acordo com demandas diversas

Cloud ML

PaaS (Platform as a Service)

- Recursos com foco na montagem e execução da aplicação
- A infraestrutura SO, hardware, servidor de aplicação, e outras tecnologias subjacentes, não controláveis pelo desenvolvedor
- Nesse contexto, a plataforma é responsável por prover suporte a recursos que a aplicação utilize

Cloud ML

SaaS (Software as a Service)

- Nesse modelo o software é usado como serviço
- O foco é apenas no uso do software ou funcionalidade
- Normalmente, referenciado como aplicações para o usuário final
  - Exemplo: serviço de email para usuário final
  - Equipe de TI não faz backup, não muda o servidor de local, não se preocupa com variação de carga, invasões, etc

Cloud ML

Estratégias de uso e montagem

- Cloud: aplicação é inteiramente montada em algum provedor de nuvem. Ou foi migrada de um ambiente local para a nuvem
- Híbrida: parte da aplicação roda localmente e parte na nuvem
  - Aumentar o número de recursos usados para executar a aplicação
- On-Premise: ambiente de nuvem montado localmente
  - Permite utilizar recursos legados com metodologias modernas de montagem de software

Montagem de Sistema

Um exemplo de PaaS é o Heroku

- Permite montar uma aplicação informando apenas os requisitos e forma de execução
- Implementa um git próprio para subir uma aplicação
- Painel na área do usuário permite executar e parar uma aplicação

Montagem de Sistema

Um exemplo de IaaS é o EC2 do AWS

- EC2 elastic compute cloud: unidade de hardware com sistema operacional e acesso por ip público
- Gerenciamento das instâncias pode ser feito pelo terminal
- É possível automatizar o processo de criar, executar uma aplicação e distribuir instâncias por scripts no terminal

Montagem de Sistema

- Um grande desafio em colocar aplicações em produção é montar o ambiente adequado de Sistema opearacional, versões de bibliotecas e parametrizações do sistema.
- Quando há duas ou mais aplicações para colocar em produção o problema se torna ainda maior, pois um único ambiente deve atender duas demandas potencialmente distintas de sistema operacional
- Uma solução para esse problema é a virtualização, que refere-se a mecanismo de criar uma visão do sistema operacional para cada aplicação

Virtualização

- Um mecanismo de virtualização muito simples são as maquinas virtuais
- Outro mecanismo mais simples os ambientes virtuais como o conda
- Duas desvantagens desses dois métodos:
  - Máquinas virtuais prejudicam o desempenho das aplicações
  - Ambientes virtuais são desprovidos de flexibilidade quanto a configuração do sistema operacional

Linux Containers (LXC)

- Sistemas operacionais modernos oferecem recursos de virtualização no nível do sistema operacional
- Tal virtualização (chamadas conteineres) parte da premissa de que o kernel do SO permite a existência de múltiplas instâncias isoladas do espaço do usuário
- Tais instâncias permitem criar um ambiente de SO próprio que acessa os recursos do computador e do SO instalado na máquina
- Do ponto de vista dos programas em execução neles, parecem computadores reais
- Conteineres são mais rápidos de iniciar, produzem pouca sobrecarga no desempenho da aplicação e são muito flexíveis quanto a montagem do Sistema operacional

Contêineres Docker

Links e Referências

- https://docker-curriculum.com/
- https://docs.microsoft.com/pt-br/visualstudio/docker/tutorials/docker-tutorial

Agenda

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

Definição

- Segregação de processos no mesmo kernel;
- Isolamento máximo possível de todo o resto do ambiente;
- File Systems, criados a partir de uma "imagem";
- Torna a reprodutibilidade muito mais fácil
- Conceitualmente semelhante a máquinas virtuais

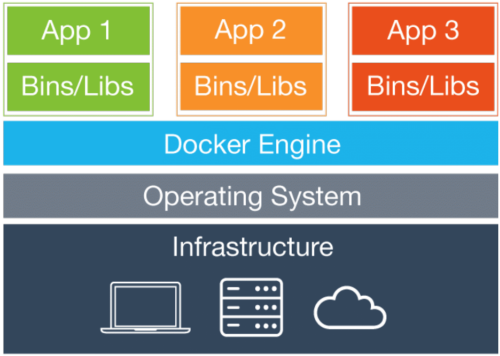


Figure: Aplicações utilizando Contêineres

Vantagens

- Leves porque não contêm um sistema operacional;
- 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 contêineres compartilham o sistema operacional do host):
  - Máquinas virtuais podem executar um sistema operacional diferente para cada máquina virtual
- Os contêineres usam isolamento no nível do processo
  - potencialmente menos seguro do que as máquinas virtuais totalmente isoladas

```
> docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:f2266cbfc127c960fd30e76b7c792dc23b588c0db76233517e1891a4
    e357d519
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
    correctly.

...

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

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

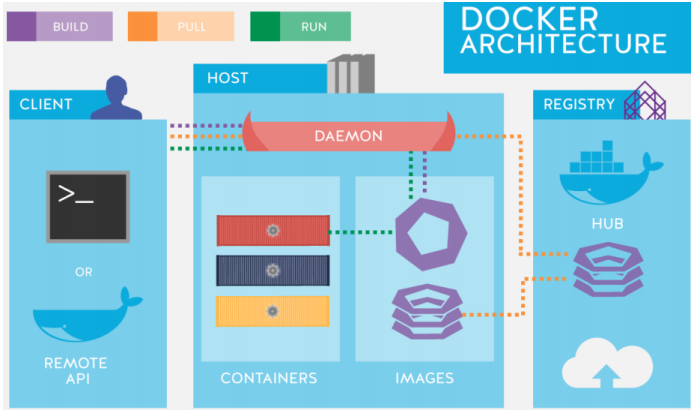


Figure: Arquitetura Docker

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: sha256:3c9c713e0979e9bd6061ed52ac1e9e1f246c9495aa063619d9d695fb
       8039aa1f
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 Contêineres
  - `docker ps -a`
- Remover Contêineres/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

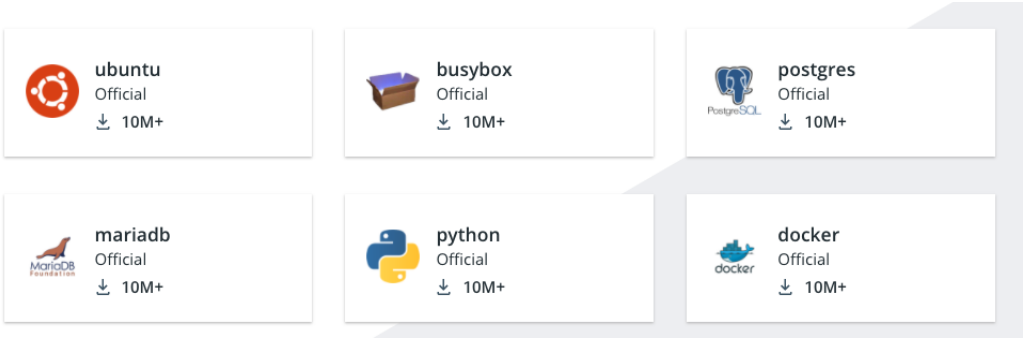


Figure: Imagens oficiais no Docker Hub

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;