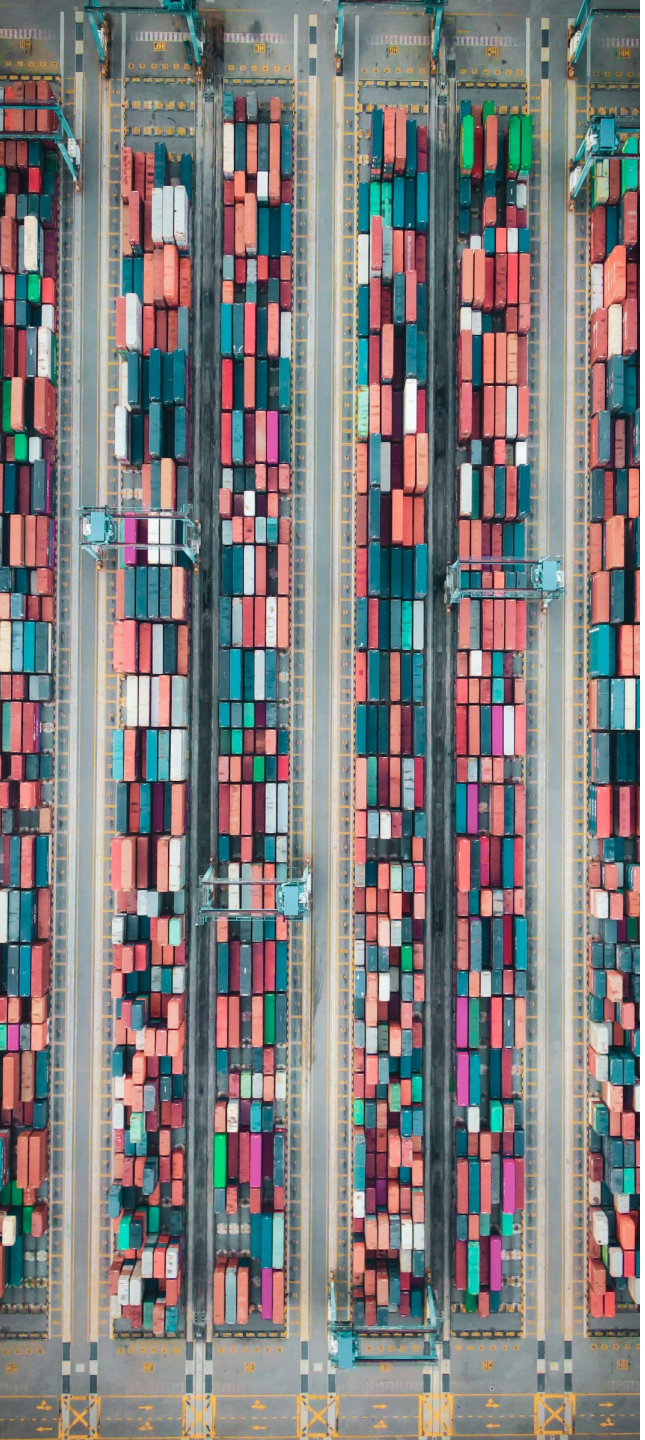


Contêinerização

Computação em Nuvem

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizado

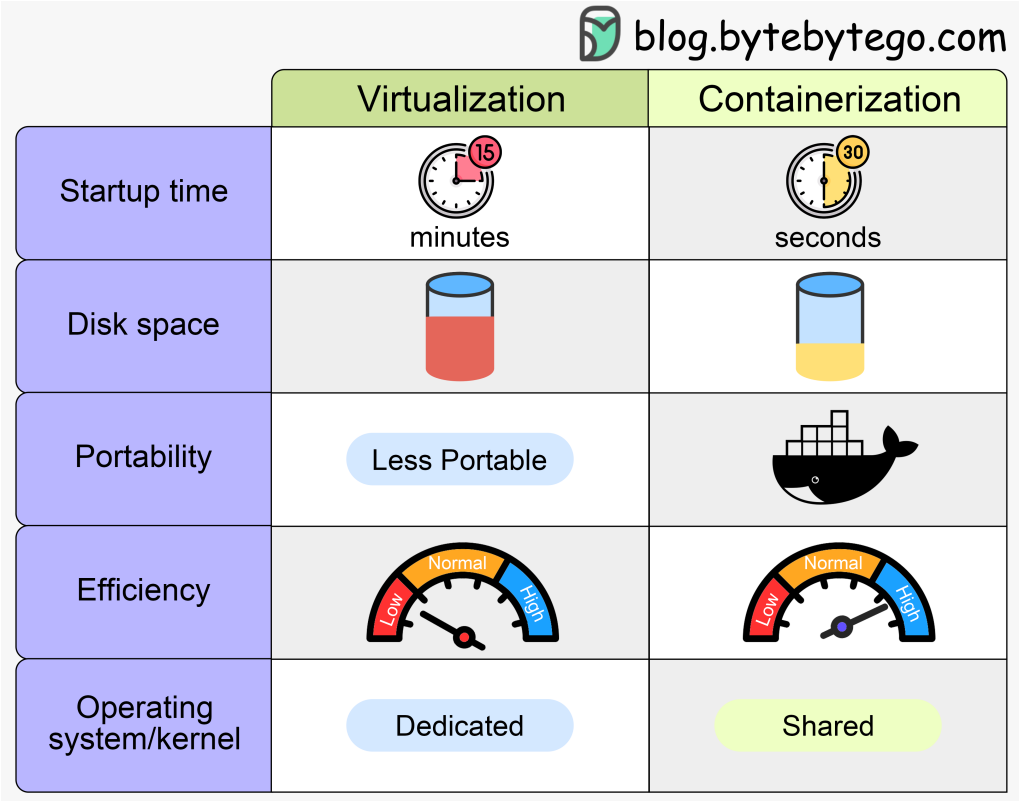
- Compreender o conceito de contêiner e o uso de aplicações conteinerizadas.
- Entender a diferença entre Máquinas Virtuais (VMs) e contêineres.
- Compreender os fundamentos da orquestração de contêineres, notadamente o Kubernetes.
- Discutir desafios práticos: desempenho, migração e gerenciamento de aplicações.

Computadores Dentro de Computadores

- Infraestrutura como Serviço (IaaS): Fornece infraestrutura virtualizada aos usuários.
- A forma mais básica de computação na nuvem é tipicamente baseada em máquinas virtuais.
  - Uma VM é a imagem de **software** de uma máquina completa que pode ser carregada em um servidor e executada como qualquer outro programa.
  - VMs permitem emular um sistema inteiramente separado dentro de um sistema existente.
  - Isolamento e Sandboxing: Essenciais para rodar código não confiável ou permitir acesso isolado a usuários não confiáveis.
- O **hypervisor** é a aplicação que supervisiona a execução das VMs, alocando e gerenciando os recursos do servidor **host**.

Contêineres: A Abordagem de Virtualização Leve

- Contêineres são frequentemente considerados uma forma de **virtualização leve**.
  - Um contêiner é basicamente um processo (ou conjunto de processos) restrito, sob o controle de um gerenciador de contêineres.
  - **Compartilhamento de Kernel:** A principal diferença é que os contêineres **compartilham o kernel** do Sistema Operacional do host.
- **Eficiência vs. Isolamento**
  - Troca **isolamento por eficiência**.
  - Contêineres são tipicamente **mais rápidos** do que VMs, pois não precisam inicializar um SO completo.



Virtualização vs. Contêinerização. Fonte: ByteByteGo.

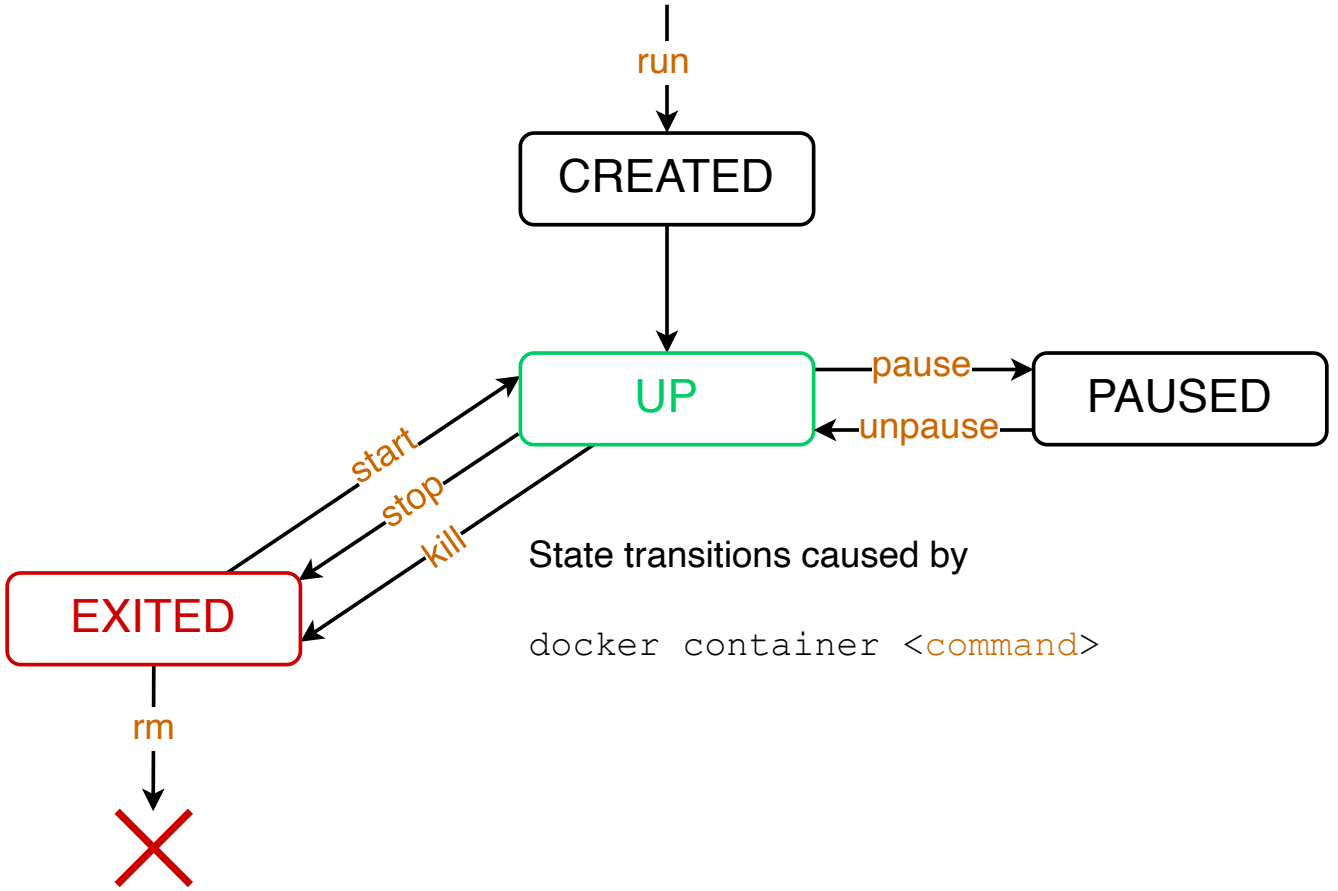
Admiral Grace Hopper

"Nos dias de pioneirismo, eles usavam bois para puxar cargas pesadas, e quando um boi não conseguia mover um tronco, eles não tentavam criar um boi maior. Não deveríamos estar buscando computadores maiores, mas sim mais sistemas de computadores."
(tradução livre do livro Cloud4Science)

Contêineres: Características do Ambiente de Execução

- **Consistência do Ambiente**: A conteinerização permite **empacotar uma aplicação** com todas as suas dependências e bibliotecas em uma única unidade de fácil gerenciamento.
  - Isso garante um **ambiente de execução consistente**.
  - O desenvolvedor pode definir: o sistema operacional esperado (embora o **kernel** seja o do **host**), as dependências e o **layout** do "disco rígido".
- Contêineres são projetados para serem **descartáveis**.
  - Após o download para o host, um contêiner pode ser iniciado quase instantaneamente (**quasi-instantly**).
  - Qualquer dado de longo prazo deve ser armazenado em **"volumes"** persistentes separados.

Container Lifecycle

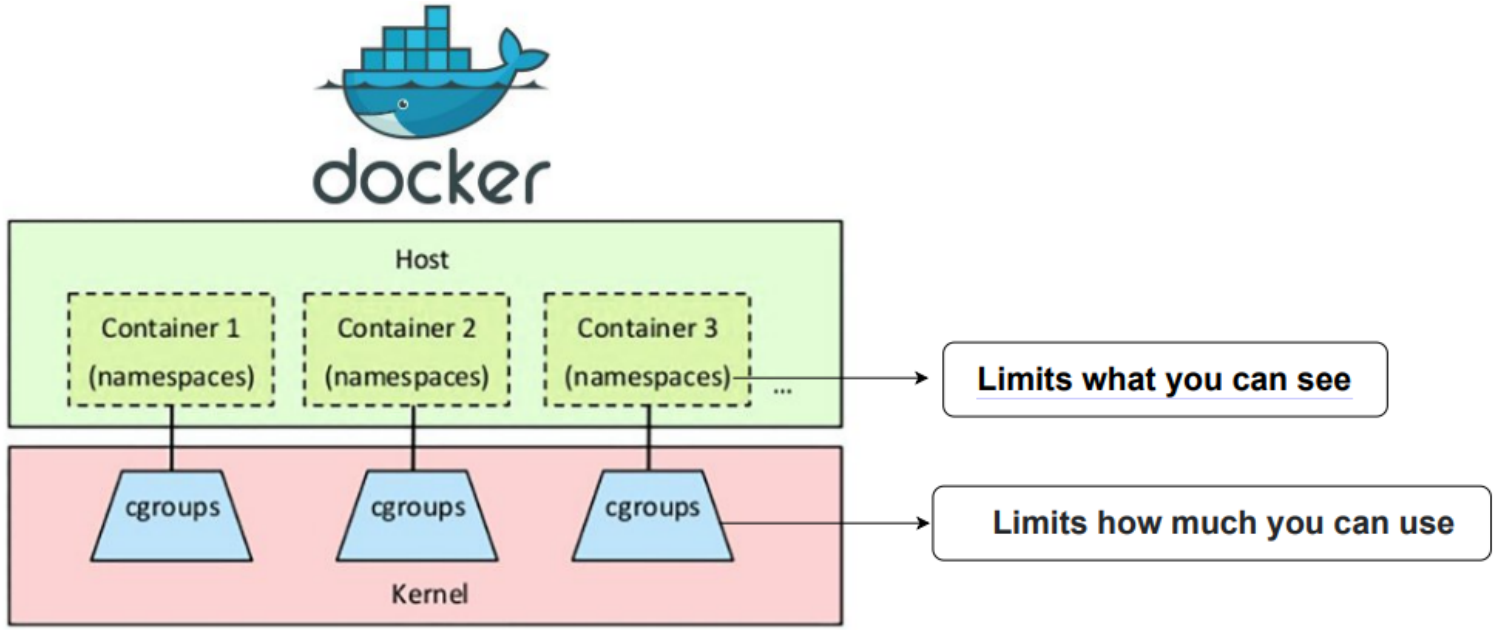


Ciclo de vida de um Contêiner. Fonte: gepardec.

Mecanismos de Isolamento (Linux)

Embora compartilhem o **kernel**, o isolamento entre contêineres é mantido através de mecanismos do SO **host**:

- **Namespaces:** Limitam o que é visível dentro do contêiner.
- **Control Groups (cgroups):** Limitam o uso de recursos (CPU, memória).



Namespace e Cgroup em Linux habilitam a criação de contêiners. Fonte: MrDevSecOps.

Gerenciamento e Construção de Contêineres

Docker é uma das ferramentas mais populares para criar e gerenciar contêineres.

1. **Imagem (Image):** modelo (blueprint) para o ambiente de execução virtualizado.
   - Cópia congelada da aplicação e de tudo o que é necessário para executá-la.
   - Exemplo analógico: Arquivo `.zip`, `.msi` ou `.dmg` que você baixa.
2. **Contêiner (Container):** Objeto efêmero, representando uma cópia do programa, baseado em uma Imagem.
3. **Dockerfile:** Script declarativo ou receita de construção (**build recipe**) usado para criar Imagens.
4. **Volumes:** Armazenam dados persistentes, garantindo que os dados não sejam perdidos após o ciclo de vida efêmero de um contêiner.

Dockerfiles: A Receita para a Imagem

São scripts usados para construir imagens, contendo instruções sobre como adicionar arquivos e dependências. Cada comando forma uma camada (layer) da imagem final.

Instrução	Descrição	Exemplo
`FROM`	Especifica a imagem base na qual a construção se apoia.	`FROM ubuntu:latest`
`RUN`	Executa comandos dados dentro de um **shell**. Estes comandos formam contêineres intermediários.	`RUN apt-get -y update`
`WORKDIR`	Especifica um novo diretório de trabalho a partir daquele ponto (como um `cd` permanente).	`WORKDIR / calculator_app`
`ADD`	Copia arquivos do diretório local (relativo ao **Dockerfile**) para dentro da imagem.	`ADD ./calculator/ ./`
`ENTRYPOINT`	Descreve o que deve acontecer quando a imagem é executada.	`ENTRYPOINT ["./ calculator"]`

Dockerfile e Arquitetura Imutável em Camadas

- Configuração orientada por **script** (script-driven configuration), facilitada pela receita **declarativa** do Dockerfile. Promove a **reprodutibilidade** do ambiente de execução.
- **Arquitetura em Camadas (Layered Architecture)**
  - **Comandos como Camadas:** Cada instrução no Dockerfile (como `RUN`), especialmente os comandos de execução, formam **contêineres intermediários**.
  - **Congelamento da Imagem:** No final do processo de construção, o Docker **"congela"** todos os contêineres intermediários em conjunto.
  - **Camadas Finais:** Cada comando, portanto, forma uma **camada** da imagem final.

Dockerfile: Copy-on-Write

- **Eficiência de Armazenamento:** O sistema de arquivos interno de um contêiner pode consistir em **múltiplas camadas** sobrepostas, utilizando o mecanismo **copy-on-write**.
  - Isso evita a cópia de todos os arquivos para novos contêineres, desde que nenhuma operação de escrita ocorra.
- A cópia dos dados só é disparada se e quando uma operação de escrita (write operation) ocorre dentro do contêiner.
  - Se um contêiner tentar modificar um arquivo que reside em uma das camadas de base, o sistema **copy-on-write** primeiro copia esse arquivo específico da camada de base para a camada de escrita (ou camada superior, exclusiva do contêiner).
  - A modificação é então aplicada a essa cópia recém-criada.
- Este mecanismo assegura que apenas as mudanças localizadas sejam armazenadas.

Dockerfile: Copy-on-Write (Demo)

Layer 1:

```
docker container run --name mod_ubuntu ubuntu:latest touch /mychange
docker container diff mod_ubuntu
```

- **Output**:
  - `A /mychange`

Dockerfile: Copy-on-Write (Demo cont.)

Layer 2:

```
docker container run --name mod_busybox_delete busybox:latest rm /etc/passwd

docker container diff mod_busybox_delete
```

- Output:
  - `C /etc`
  - `D /etc/passwd`

Dockerfile: Copy-on-Write (Demo cont.)

Layer 3:

```
docker container run --name mod_busybox_change busybox:latest touch /etc/passwd

docker container diff mod_busybox_change
```

- Output:
  - `C /etc`
  - `C /etc/passwd`

Orquestração de Contêineres

- **Orquestração de Contêineres**: Quando um sistema possui muitos contêineres, é necessário um **meta-sistema** para gerenciar, coordenar e monitorar os contêineres individuais.
- **Funcionalidades Típicas de um Orquestrador:**
  - **Controle de Limite de Recursos:** Definição de restrições para reservas de CPU ou memória para os contêineres.
  - **Scheduling** (Agendamento): Determina qual contêiner (ou **Pod** no Kubernetes) será atribuído a qual máquina no **cluster**.
  - **Balanceamento de Carga (Load Balancing):** Distribuição de requisições de entrada entre múltiplas instâncias de contêiner.
  - **Tolerância a Falhas e Verificação de Saúde (Health check):** Detecção de falhas e substituição de contêineres defeituosos para manter um número pré-definido de réplicas saudáveis.
  - **Autoscaling**: Adicionar e remover contêineres automaticamente dependendo da carga atual.

Kubernetes: Datacenter as a Service

- Kubernetes (K8s) é um dos orquestradores de contêineres mais populares e dominantes.
  - Mantido pela Cloud Native Computing Foundation (CNCF).
  - Informalmente referido como "Datacenter as a Service".
- Usuários do Kubernetes descrevem declarativamente clusters inteiros, incluindo abstrações para computação, armazenamento e rede.
- A configuração é geralmente mantida em arquivos de texto simples usando a sintaxe YAML.
- Conceitos Fundamentais (Recursos):
  - Pod: A unidade básica de scheduling e deployment no Kubernetes. É uma coleção de contêineres que são iniciados e parados juntos na mesma máquina (Node).
  - Deployment: Um objeto de API que gerencia Pods, incluindo a definição do número de réplicas desejadas para escalonamento horizontal.
  - Service: Uma abstração que expõe uma aplicação de rede (um ou mais Pods), atuando como balanceador de carga.

Contêineres e a Computação em Nuvem

- Provedores de Nuvem oferecem diversas formas de usar contêineres:
  - Virtual Private Server (VPS): Basicamente uma VM.
  - Serviços de Execução de Aplicações (App Runner/Apps): O provedor se oferece para executar seu contêiner em seus servidores.
  - Serviços Kubernetes: Implantações de K8s, frequentemente com integrações específicas da plataforma de **cloud**.
- DevOps e Automação
  - A conteinerização, combinada com sistemas de controle de versão (como Git), é um facilitador das práticas DevOps.
  - Objetivo: Reduzir a lacuna entre desenvolvimento e operações, buscando ciclos de release rápidos e alto grau de automação e estabilidade (CI/CD).
  - Infraestrutura como Código (IaC): Princípios que definem configurações de infraestrutura usando arquivos declarativos, simplificando a implantação.

Considerações Finais

- A Essência da Contêinerização: Contêineres são processos em execução, definidos por imagens, que oferecem uma duplicação eficiente e isolada de ambientes de execução.
- Gerenciamento de Escala: Em arquiteturas modernas de software, é comum a presença de inúmeros contêineres. O orquestrador Kubernetes é a solução dominante para o gerenciamento desses contêineres.
- Implicações para Ciência de Dados: O uso de contêineres garante que as ferramentas, bibliotecas e ambientes de execução para modelos e análises de dados sejam reprodutíveis e portáveis, evitando o problema da "síndrome do funciona na minha máquina
- Leituras recomendadas: Google Cloud: O que é conteinerização? || Cloud4Science, capítulo 6 || Research for practice: the DevOps phenomenon

Perguntas e Discussão

- Qual é a principal diferença arquitetônica entre Máquinas Virtuais (VMs) e Contêineres em termos de uso do **Kernel** do Sistema Operacional? Como essa diferença fundamental influencia diretamente no **trade-off** entre **eficiência de inicialização** (velocidade) e **nível de segurança e isolamento** fornecido ao sistema **host**?
- Em um projeto de Ciência de Dados (onde a reprodutibilidade é vital), de que maneira a definição declarativa de um ambiente por meio do `Dockerfile` permite que um modelo de AI/ML opere conforme projetado?
- Explique o conceito de um **Dockerfile** e como a arquitetura arquitetura de camadas (**layers**), combinada com o mecanismo de **copy-on-write**, permite que novos contêineres sejam iniciados **quase instantaneamente**.
- Qual a maior vantagem de utilizar contêineres ao invés de máquinas virtuais no deploy de aplicações em múltiplos provedores de cloud diferentes?