5954024 - Introdução ao Desenvolvimento Web

Introdução ao Docker para WebDev

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

1. Compreender o conceito de contêiner e o uso de **aplicações conteinerizadas**.
2. Entender a **diferença** entre Máquinas Virtuais (VMs) e contêineres.
3. Discutir desafios práticos: desempenho, migração e gerenciamento de aplicações.
4. Implementar soluções Web com contêiners.

> Objetivos de 1 a 3 cobertos na Prova Teórica.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Motivação

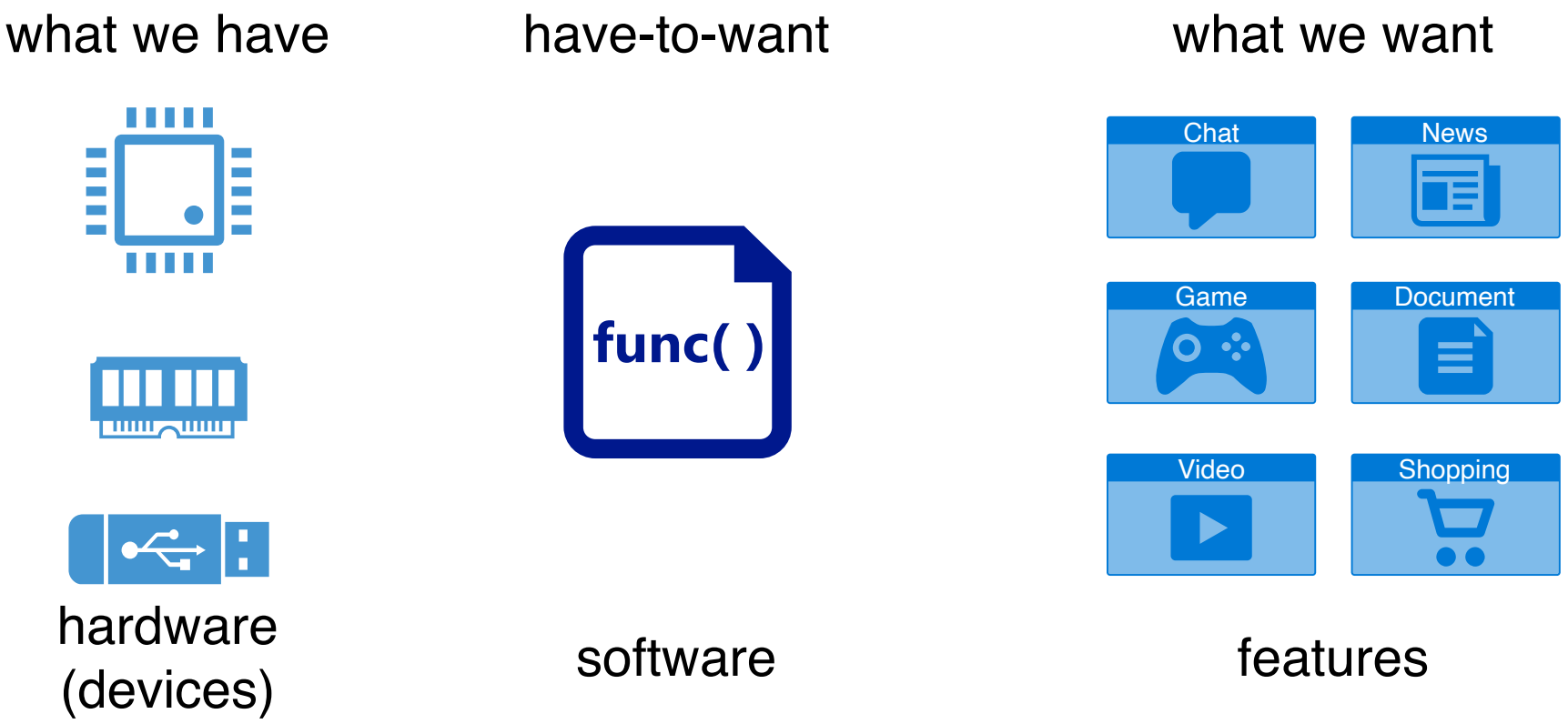
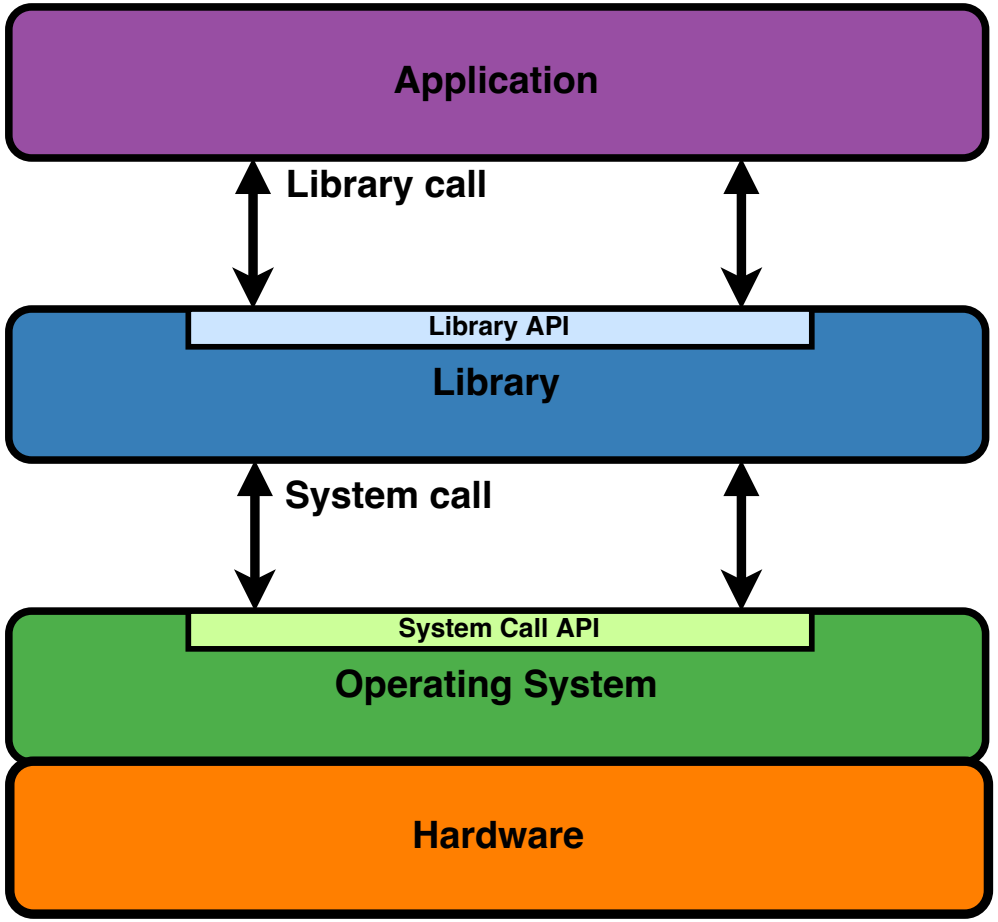


Fig. .1: Fonte da Imagem: https://open-education-hub.github.io/operating-systems/

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Relembrando: Sist. Operacionais

- SO é um **software** que:
  - Utiliza recursos de hardware de um sistema computacional, e
  - Provê suporte para execução de outros software.
- SO atua como intermediário entre o hardware e os programas do usuário.
- Garante a execução eficiente e segura de múltiplos processos.

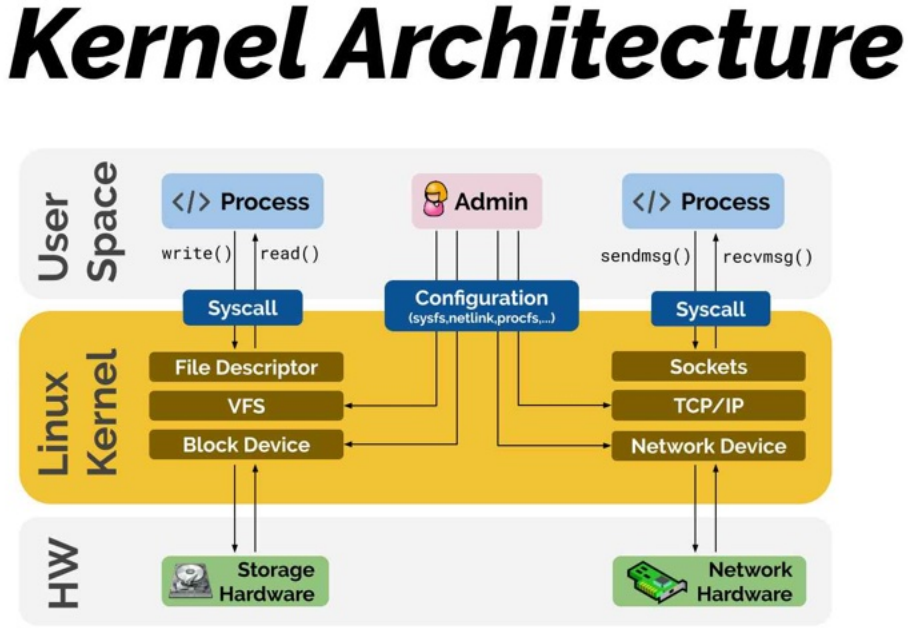


Software Stack. Fonte da Imagem: OS OER.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Relembrando: Kernel de um SO

- **Função geral**: Gerenciar todos os recursos de hardware: CPU, memória e E/S (I/O).
- Fornecer um conjunto de APIs, independentes da arquitetura e do hardware, para permitir que aplicações e bibliotecas no espaço do usuário utilizem os recursos de hardware.
- Lidar com acessos e uso concorrentes dos recursos de hardware por diferentes aplicações.



> Na imagem: Arquitetura geral do Kernel Linux.
>
> Fonte da Imagem: eBPF - Rethinking the Linux Kernel

Admiral Grace Hopper

"(...) 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."

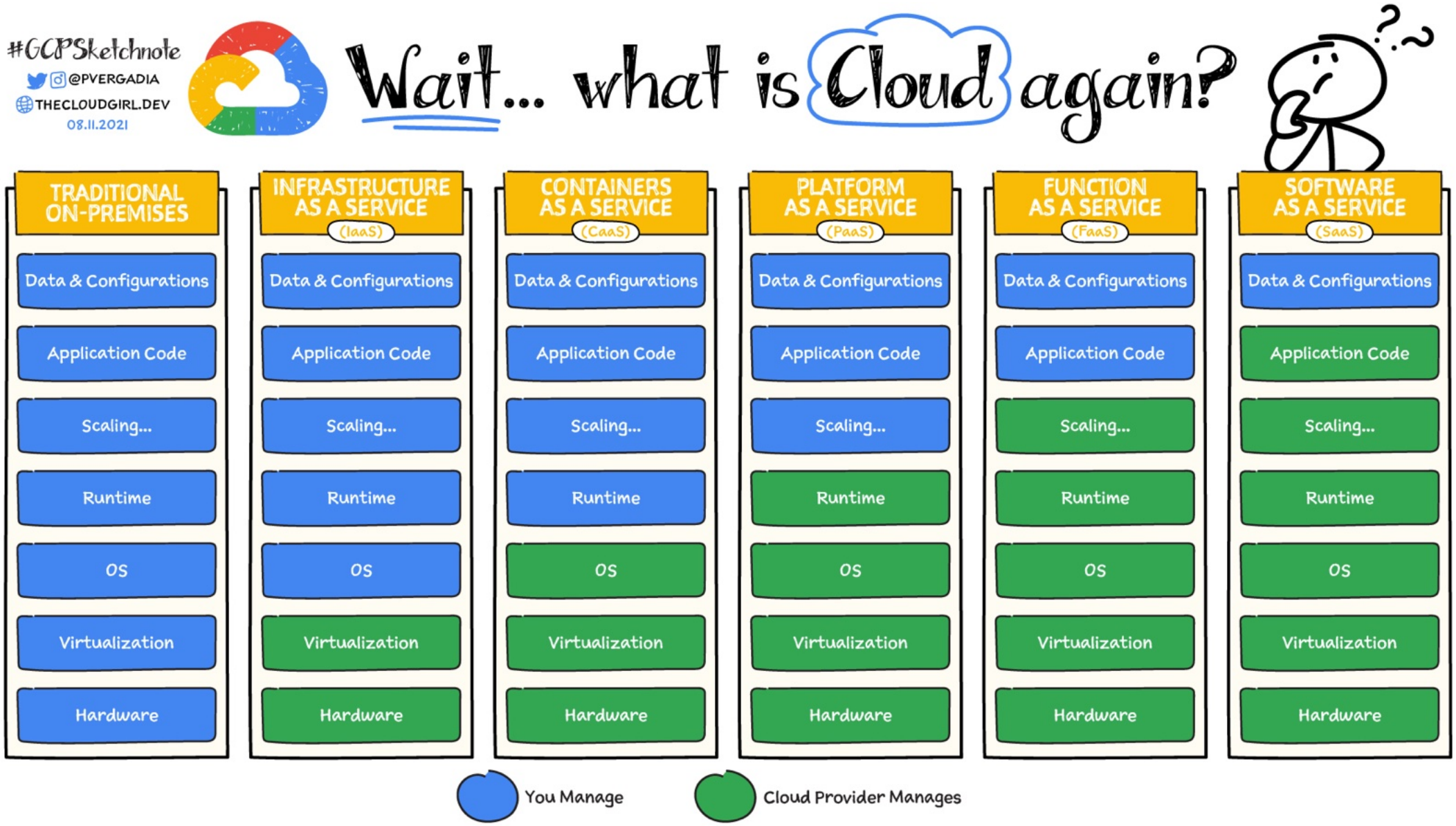
> Tradução livre de trecho do livro Cloud4Science

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Contextualizando: Modelos de Serviço/Entrega em Nuvem

- Descrevem maneiras de disponibilizar recursos de TI sob demanda na nuvem.
- Os pilares são: Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS), Software como Serviço (SaaS), Função as a Service (FaaS) Anything as a Service (XaaS).
- Cada um confere ao usuário níveis variados de controle, flexibilidade e responsabilidade compartilhada entre cliente e provedor.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br


#GCPSketchnote
@PVERGADIA
THECLOUDGIRL.DEV
08.11.2021
Wait... what is Cloud again?
TRADITIONAL ON-PREMISES
INFRASTRUCTURE AS A SERVICE (IaaS)
CONTAINERS AS A SERVICE (CaaS)
PLATFORM AS A SERVICE (PaaS)
FUNCTION AS A SERVICE (FaaS)
SOFTWARE AS A SERVICE (SaaS)
Data & Configurations
Application Code
Scaling...
Runtime
OS
Virtualization
Hardware
You Manage
Cloud Provider Manages

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*.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Virtualização

- Permite criar diversos servidores virtuais em um único servidor físico.
- Hypervisor/Virtual Machine Monitor (VMM): Responsável por criar e executar VMs. Gerencia a alocação de CPU, memória, dispositivos de E/S e rede.

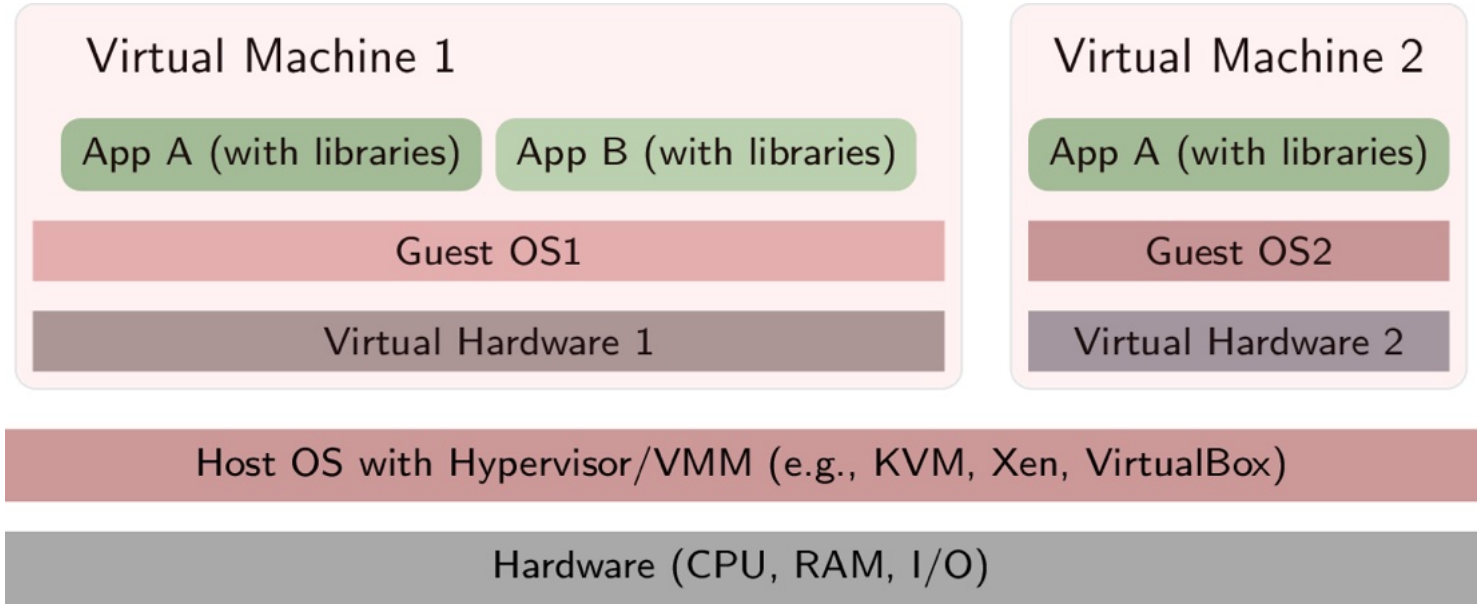


Fig. .1: Virtualização por Jens Lechtenboerger.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Contêineres: Abordagem de Virtualização Leve

- Um contêiner é basicamente um conjunto de processos restrito, sob o controle de um gerenciador de contêineres.
- Contêineres **compartilham o kernel** do Sistema Operacional do host.
- **Eficiência vs. Isolamento**: Contêineres são tipicamente **mais rápidos** do que VMs, pois não precisam inicializar um SO completo.

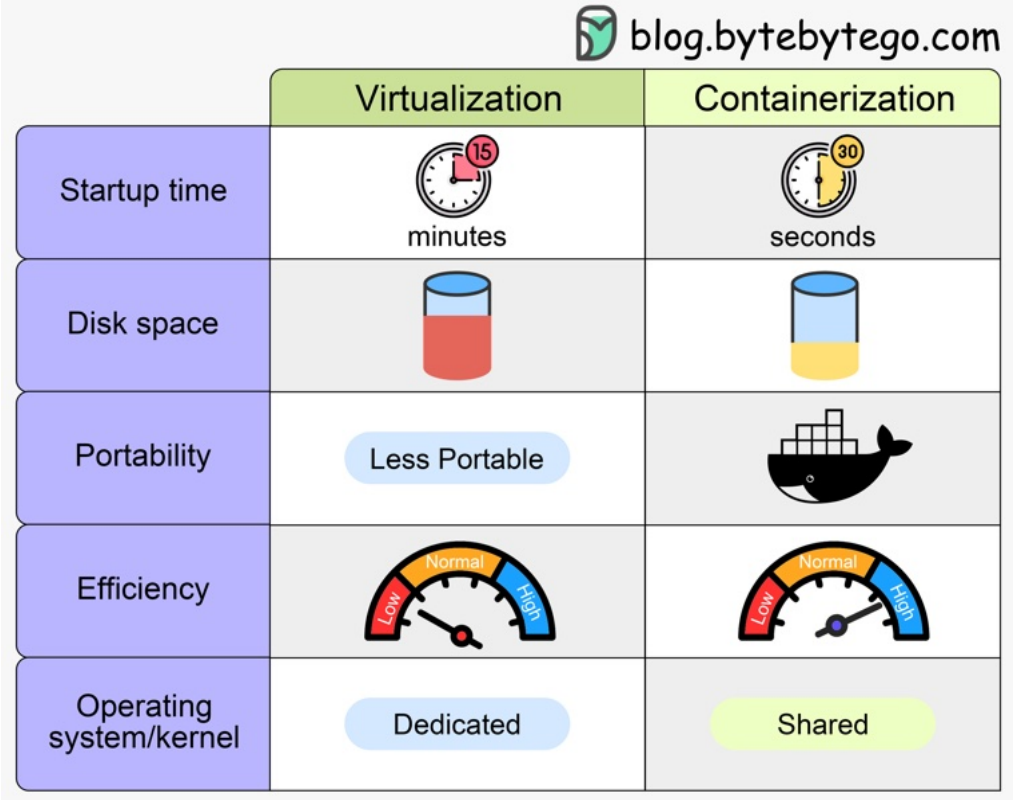


Fig. .1: Virtualização vs. Contêinerização por ByteByteGo.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Gerenciamento e Construção de Contêineres

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

- **Imagem (Image):** modelo para o ambiente de execução virtualizado. Cópia congelada da aplicação e de tudo o que é necessário para executá-la.
- **Contêiner (Container):** Objeto efêmero, representando uma cópia do programa, baseado em uma Imagem.

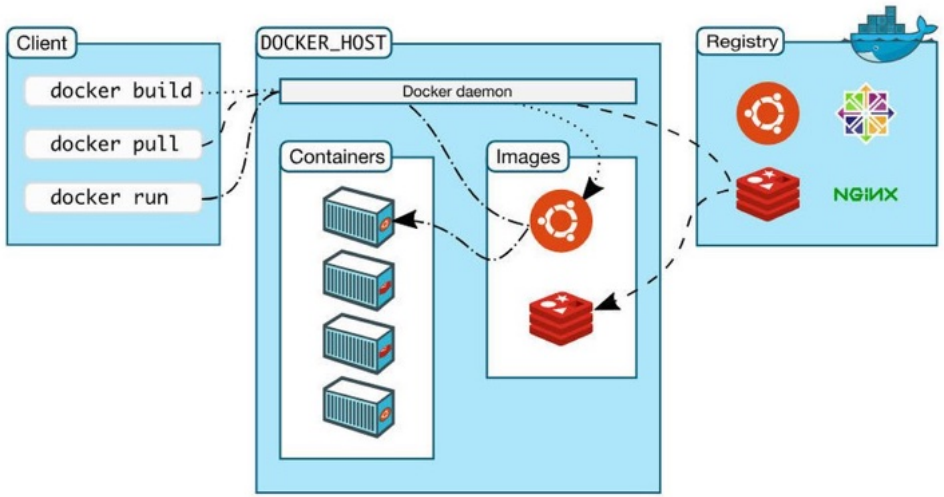


Fig. .1: Ecossistema Docker por Data Center Insider.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Nginx

- Um servidor HTTP de alto desempenho.
- Consome menos recursos que um servidor Apache tradicional.

Comando:

```
docker run nginx
```

- **docker**: Executa o comando Docker.
- **run**: Cria e inicia um container novo.
- **nginx**: Nome da imagem oficial no Docker Hub.

> O container roda em modo **foreground** (bloqueia seu terminal). Ao fechar o terminal, o container para.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Mapeamento de Portas (`-p`)

Para acessar o site via navegador, precisamos conectar a porta do container à sua máquina.

```
docker run -p 8080:80 nginx
```

- **`-p`**: Flag de mapeamento de portas.
- **`8080`**: Porta na sua máquina (Host).
- **`80`**: Porta padrão do Nginx dentro do container (Container).

Resultado: Ao acessar http://localhost:8080, você verá a página padrão do Nginx.

> **Nota:** Nunca exponha a porta 80 diretamente em produção sem HTTPS. Use 443 para SSL/TLS.

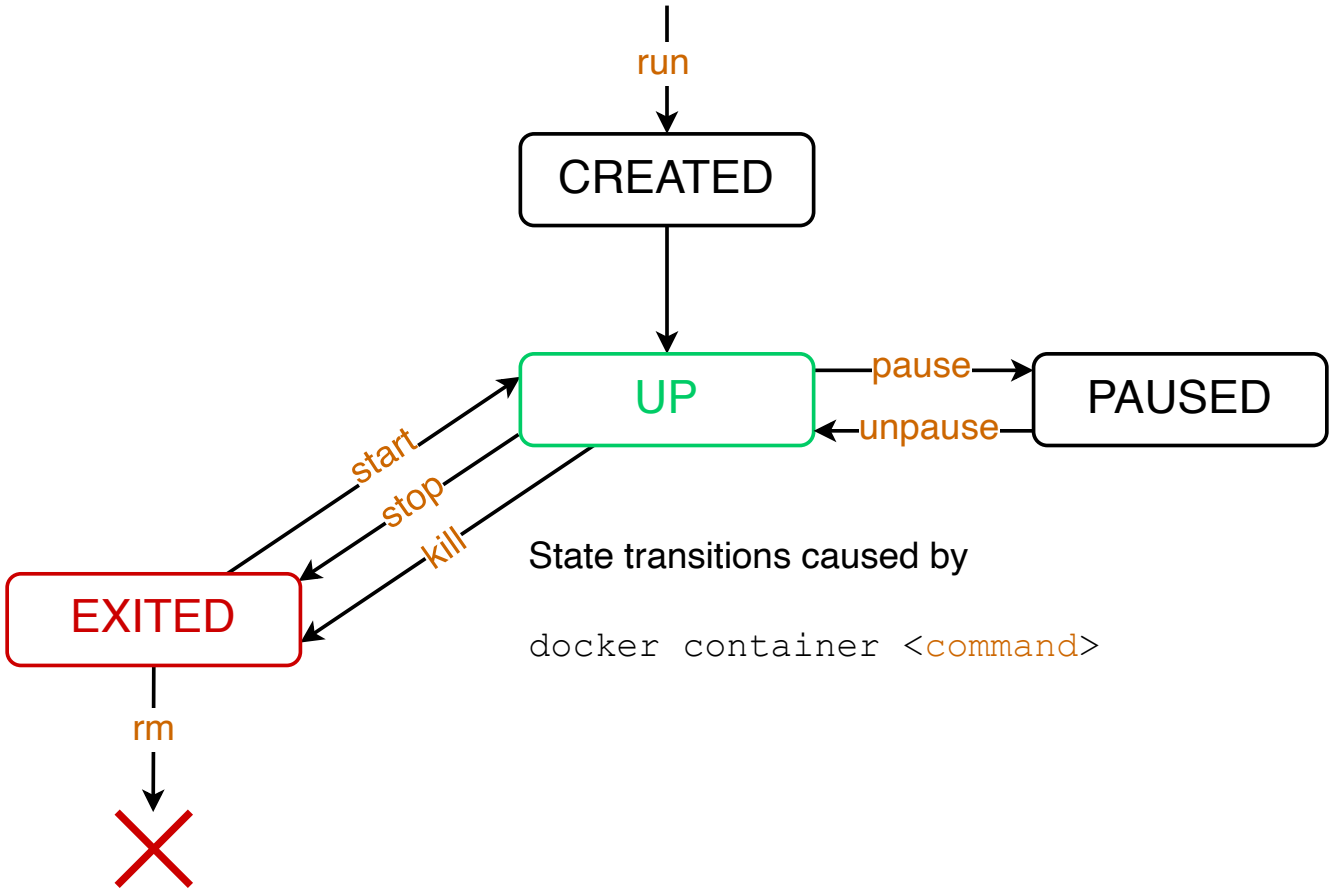
Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

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.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Lifecycle



Ciclo de vida de um Contêiner. Fonte: gepardec.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Persistência de Dados com Volumes (`-v`)

```
docker run -p 8080:80 \
  -v /home/aluno/projetos/web:/usr/share/nginx/html \
  nginx
```

- **`-v`**: Flag de volume (mount).
- **`/home/aluno/projetos/web`**: Caminho no seu computador (Host).
- **`/usr/share/nginx/html`**: Caminho padrão onde o Nginx procura arquivos.

Crie um arquivo **`index.html`** na pasta do projeto e ele aparecerá automaticamente no site sem precisar reconstruir o container.

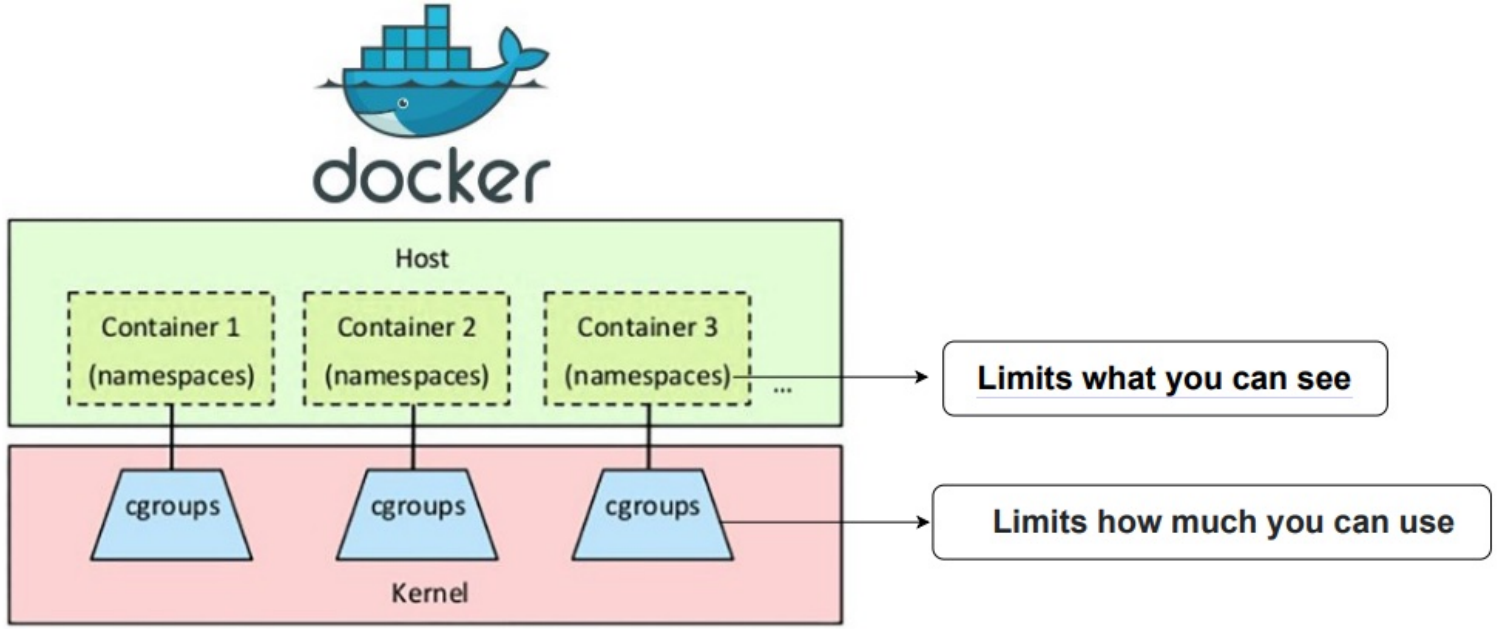
> Contêineres são projetados para serem **descartáveis**. Se você deletar um contêiner, os dados somem.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

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).



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Gerenciamento e Nomeação

Rodando em Background: Não queremos travar o terminal. Vamos nomear e rodar no plano de fundo.

```
docker run -d \
  --name web-nginx-local \
  -p 8080:80 \
  -v /home/aluno/projetos/web:/usr/share/nginx/html \
  nginx
```

Novos Parâmetros:

- **-d**: Run em modo **detached** (background). O terminal não trava.
- **--name**: Dá um nome ao container para facilitar o controle (**docker ps**).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Comandos Úteis

- **docker ps -a**: Ver containers rodando.
- **docker logs web-nginx-local**: Ver os logs do servidor.
- **docker stop web-nginx-local**: Parar o servidor.
- **docker start web-nginx-local**: Reinicia o container.
- **docker rm web-nginx-local**: Remove o container (após parar).

Posso remover um container que está rodando?

Não. Primeiro pare-o com **docker stop**, depois remova com **docker rm**.

> O comando **docker rm** é destrutivo. Se você remover um container, perde o acesso à sua instância atual. Em produção, use volumes para persistir dados antes de remover.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dockerfile

- **Script Declarativo** que atua como uma **receita de construção** (*build recipe*) para a criação de Imagens Docker.
- **Arquitetura em Camadas (*Layers*):** Cada instrução contida no Dockerfile gera uma nova camada imutável na imagem final. Eficiência com Copy-on-Write.
- Garante que o ambiente de execução seja idêntico em qualquer estágio (desenvolvimento, teste ou produção).
- Evita que você ganhe o certificado Works on My Machine.

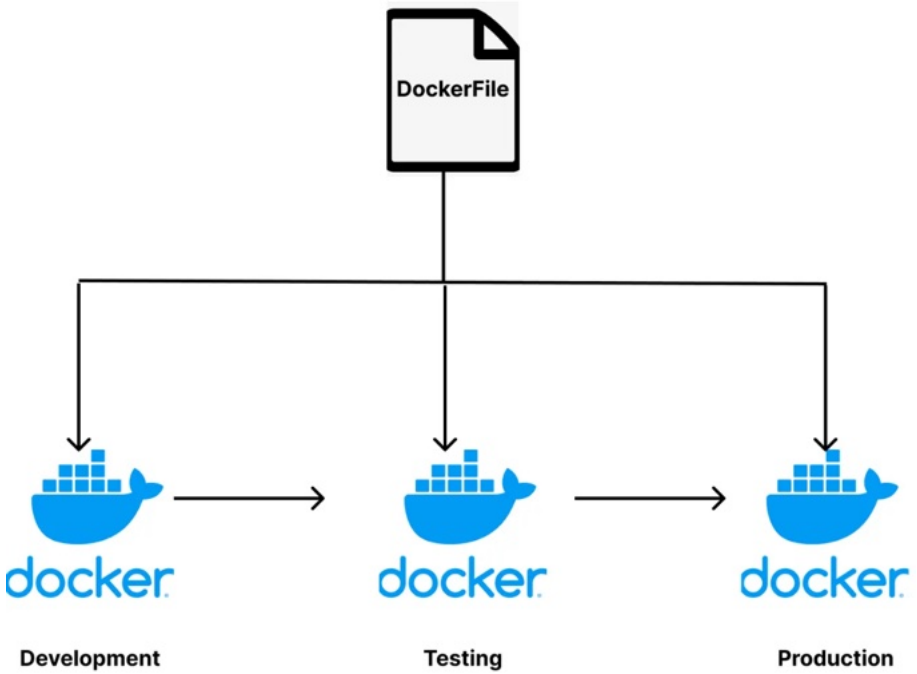


Fig. .1: Dockerfile e multistage por Ashish Singh.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Dockerfile (cont.)

Antes de criar o Dockerfile, precisamos organizar nossa pasta:

```
meu-site/
├── css/
│   └── style.css
├── js/
│   └── app.js
├── Dockerfile
└── index.html
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Dockerfile (cont.)

```
# Define a imagem base
FROM nginx:alpine
# Define um diretório de trabalho
WORKDIR /usr/share/nginx/html
# Copia arquivos para o container
COPY index.html .
COPY css/ ./css/
COPY js/ ./js/
# Define a porta exposta (não é mapeamento)
EXPOSE 80
# Comando inicial
CMD ["nginx", "-g", "daemon off;"]
```

> Cada instrução (FROM, RUN, COPY) cria uma camada na imagem, otimizada pelo mecanismo Copy-on-Write.

Dockerfile: Copy-on-Write

- 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.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Copy-on-Write

Layer 1:

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

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

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Copy-on-Write (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**

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Copy-on-Write (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**

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

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 atualmente dominante para o gerenciamento desses contêineres.
- **Leituras recomendadas**:
  - Google Cloud: O que é conteinerização?
  - Research for practice: the DevOps phenomenon

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício e Questões

Exercício Prático

Você desenvolveu um portfólio pessoal simples, composto apenas por arquivos HTML e CSS estáticos.

Tarefa:

1. Crie uma pasta chamada **portfolio_static**. Dentro dela, coloque seus arquivos (**index.html**, **style.css**, etc.).
2. **Crie um arquivo chamado** **Dockerfile** (sem extensão) dentro desta pasta. Este arquivo deve conter as instruções para o Docker construir a imagem do seu site.
3. O container final deverá expor a porta 80 e ser acessível no seu navegador após rodar o comando de execução.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para Estudo

- 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**?
- 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**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para Estudo (cont.)

Imagine que você está desenvolvendo uma API RESTful simples usando Python/Flask. Descreva, passo a passo, o ciclo de vida completo dessa aplicação desde o código fonte até estar rodando em um container isolado. Sua explicação deve cobrir os seguintes elementos:

1. **O Papel do `Dockerfile`:** Explique como as instruções (`FROM`, `RUN`, `COPY`, `CMD`) guiam a construção da imagem. Dê um exemplo de por que o comando `WORKDIR` é crucial.
2. **A Imagem (Image):** O que exatamente uma "imagem" representa no contexto do Docker? É apenas um arquivo ZIP? Justifique sua resposta.
3. **O Container:** Qual a relação entre a imagem e o container? Explique como o conceito de *Volumes* resolve esse problema em aplicações web que precisam persistir estado (ex: banco de dados).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br