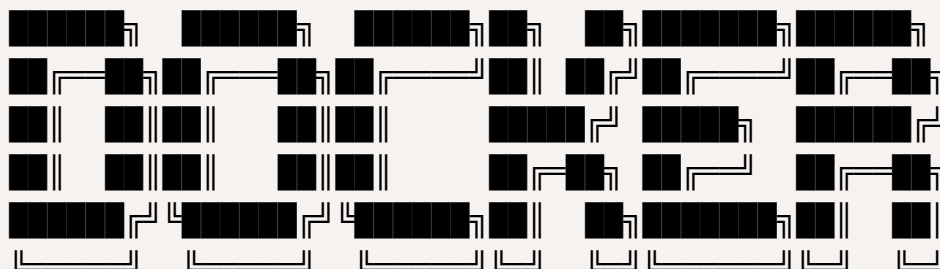




# Table of Contents

Bem-vindo ao Learn Docker! 🐳	2
Por que Docker	7
Containers vs Máquinas Virtuais	12
História do Docker	22
Instalação do Docker 🛠️	29
Docker Desktop vs CLI: Escolha sua Arma 🚀	30
Problemas Comuns: Debugando na Matrix 🐍	35
Containers 101	41
Container Internals: Como Funciona Por Dentro 🔬	64
Container Runtime: O Motor dos Containers 🚀	74
Segurança de Containers 🛡️	89
Orquestração de Containers: A Sinfonia do Caos Digital	100
Container Monitoring: Os Olhos do Sistema 👁️	109
Images vs Containers: O Dual Boot do Futuro 🖥️	119
Fundamentos de Imagens Docker 📦	124
Anatomia de uma Imagem Docker 🔬	130
Gerenciamento de Imagens Docker 🎮	136
Operações com Containers: O Manual do Operador 🎮	142
Melhores Práticas Docker: O Caminho do Mestre 🎯	149
Laboratório Prático: Mão na Massa! 🧪	156
Guia de Troubleshooting: Debugando na Matrix 🔍	163
Arquitetura Docker: O Blueprint da Matrix 🏗️	169
CLI Basics: Dominando o Terminal Docker 🎮	176
Docker Commands: O Arsenal Completo 🚀	183
Docker Images: Blueprint do Seu Container 🎨	190
Ciclo de Vida dos Containers: Do Nascimento à Transcendência ↺	201
Container Networking	211
Container Storage: Crônicas dos Dados Perdidos 💾	223

# Bem-vindo ao Learn Docker!



CONTAINERS AS CODE = LIFE

[ CYBERPUNK EDITION 2025 ]

BUILD ONCE

RUN ANYWHERE

SHIP FASTER

## Sobre este Guia

Bem-vindo ao guia mais irreverente e prático sobre Docker! Este não é apenas mais um tutorial chato - é sua jornada cyberpunk pelo mundo dos containers.

## O que você vai encontrar aqui

 Estrutura do Conteúdo

- **INIT: Container Matrix** - Sua pílula vermelha para o mundo Docker
- **BASICS: Docker Fundamentals** - Aprenda a hackear o mainframe (ou pelo menos rodar um container)
- **ADVANCED: Docker Deep Dive** - Entre na toca do coelho
- **PRACTICE: Hands-on Labs** - Porque teoria sem prática é como um container sem imagem
- **CERTIFICATION: Docker Certified Associate** - Torne-se um operador certificado da Matrix

## **Como Usar este Guia**

1. **Siga a Ordem:** Cada seção foi cuidadosamente planejada para construir seu conhecimento gradualmente
2. **Pratique Muito:** Cada conceito vem com exercícios práticos
3. **Divirta-se:** Sim, aprender Docker pode (e deve) ser divertido!

## **Recursos Especiais**

- **Waifu Tech Squad:** Suas guias pelo mundo dos containers
- **ASCII Art:** Porque documentação técnica precisa de arte
- **Easter Eggs:** Encontre-os todos!
- **Referências Pop:** De Matrix a Blade Runner, tudo está aqui

## **Pré-requisitos**

- Um computador (duh!)
- Terminal básico
- Vontade de aprender
- Senso de humor

- Tolerância a referências geek

## Quick Start

```
# Clone este repositório
git clone ${project_repo}

# Entre na pasta do projeto
cd learn-docker

# Comece sua jornada
./start-learning.sh
```



## Quick Reference

```
COMANDOS ESSENCIAIS

docker run    → Criar e iniciar container
docker ps     → Listar containers
docker build  → Construir imagem
docker pull   → Baixar imagem do registry
docker push   → Enviar imagem para registry
```

|| ARQUIVOS IMPORTANTES

||

||

==||

|| Dockerfile → Receita para construir imagens

||

|| docker-compose → Orquestração de múltiplos containers

||

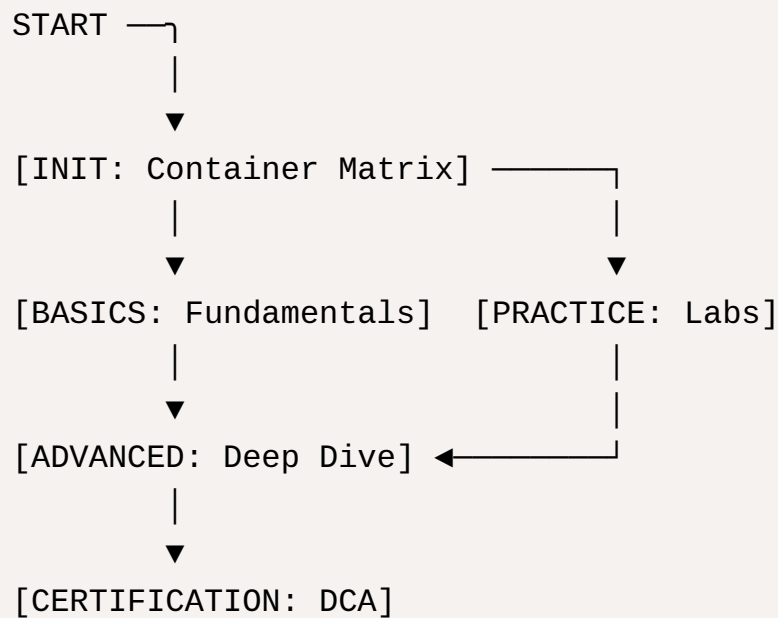
|| .dockerignore → Arquivos a serem ignorados no build

||

||

==||

## Mapa de Navegação



## Contribuindo

Encontrou um bug? Quer adicionar mais referências cyberpunk? Abra uma issue ou PR!

## Código de Conduta

1. Não fale sobre o Fight Club

2. Containers são amigos, não comida
3. Sempre cite suas referências geek
4. Respeite as waifus

## Contato

- **Discord:** [Link para o servidor]
- **Twitter:** [@DockerMaster]

## Licença

Este projeto está licenciado sob a Licença MIT - veja o arquivo para detalhes.



"Eu posso te mostrar a porta, mas você é quem tem que atravessá-la."

- Morpheus (provavelmente falando sobre Docker)

Pronto para começar? para ver o roadmap completo!

# Por que Docker

```
|| W H Y   D O C K E R ?
```

```
|| "Porque funciona na minha máquina" não é mais uma desculpa  
válida ||
```

## TL;DR (Too Long; Didn't Read)

```
| Docker = Containers |  
| Containers = 📦      |  
| 📦 = Portabilidade  |
```

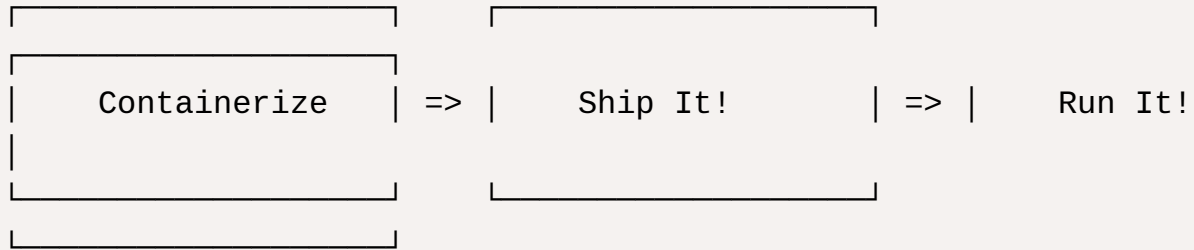
## O Problema 🔥

Já se perguntou por que:

- Seu código funciona localmente mas quebra em produção?
- Configurar um novo ambiente de desenvolvimento é um pesadelo?
- Cada desenvolvedor da equipe tem uma versão diferente das dependências?
- O novo dev levou 3 dias só para configurar o ambiente?



## A Solução



Docker resolve estes problemas através de **containers**: ambientes isolados e portáteis que:

- Empacotam aplicação + dependências
- Garantem consistência entre ambientes
- São leves e rápidos de iniciar
- Facilitam o deploy

## Vantagens Reais

### 1. Consistência

Dev: "Funciona na minha máquina!"  
Ops: "Agora funciona em todas!"

### 2. Isolamento

- Cada container é um ambiente isolado
- Sem conflitos entre versões
- Segurança melhorada

### 3. Portabilidade

- Rode em qualquer lugar
- Linux, Windows, Cloud
- Mesmo comportamento garantido

## 4. Eficiência 🚀

- Inicialização em segundos
- Menos recursos que VMs
- Escalabilidade facilitada

## Casos de Uso 🎮

### Desenvolvimento

```
$ docker-compose up  
> ambiente pronto! 🎉
```

### Testes

```
$ docker test  
> testes consistentes 🧪
```

### Produção

```
$ docker deploy  
> deploy confiável 🚀
```

## Docker vs VMs 🥊

Docker	VMs
▲ Mais leve	▼ Mais pesadas
▲ Boot em segundos	▼ Boot em minutos
▲ Menos recursos	▼ Mais recursos
▲ Compartilha OS	▲ Mais isolamento

## Começando 🎬

### Requisitos Mínimos

- Um computador (duh!)
- Sistema operacional (Windows/Linux/Mac)
- Vontade de aprender
- Café (muito café)

### Próximos Passos

1. Instalação do Docker ([Instalação do Docker](#) 🛠️)
2. Conceitos Básicos ([Containers 101](#))
3. Primeiro Container ([Docker Commands: O Arsenal Completo](#) 🚀)

## Waifu Tips 💡


⚠️ **Container-chan diz:** "Lembre-se: containers são efêmeros! Não se apegue a eles como se apegue ao seu waifu body pillow!"

## Checkpoint

Você agora sabe:

- [x] Por que Docker é importante
- [x] Principais benefícios
- [x] Diferença entre Docker e VMs
- [x] Casos de uso comuns

## Exercícios

1. Liste 3 problemas que você já enfrentou que poderiam ser resolvidos com Docker
2. Pesquise uma empresa que usa Docker e como ela se beneficia
3. Tente explicar Docker para seu rubber duck 



"A jornada de mil containers começa com um único `docker run` "

- Container-chan, 2025

# Containers vs Máquinas Virtuais

C O N T A I N E R S    v s    V M S

Round 1: FIGHT!

💡 Containers vs VMs

💥 Quem vence essa batalha?

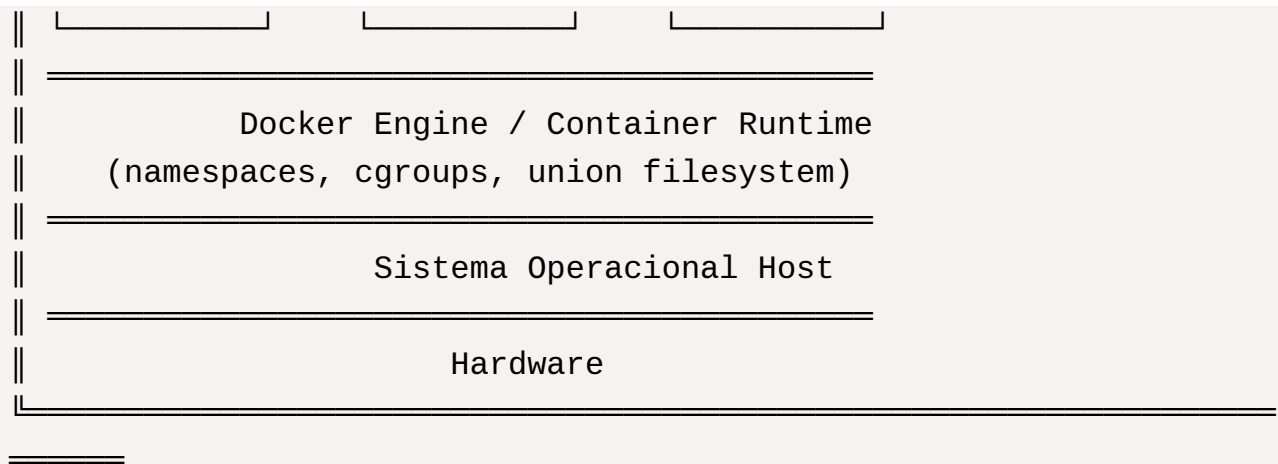
## TL;DR (Too Long; Didn't Read)

Containers	VMs
▲ Leves	▼ Pesadas
▲ Boot: segundos	▼ Boot: minutos
▲ MB de tamanho	▼ GB de tamanho
▼ Menos isolado	▲ Mais isolado

## Anatomia Detalhada

### Containers em Profundidade

Container A	Container B	Container C
App A	App B	App C
Bins	Bins	Bins
Libs	Libs	Libs



## Componentes do Container

### 1. Namespaces

- PID: Isolamento de processos
- NET: Isolamento de rede
- MNT: Pontos de montagem
- UTS: Hostname e domínio
- IPC: Comunicação interprocessos
- USER: IDs de usuários

### 2. Control Groups (cgroups)

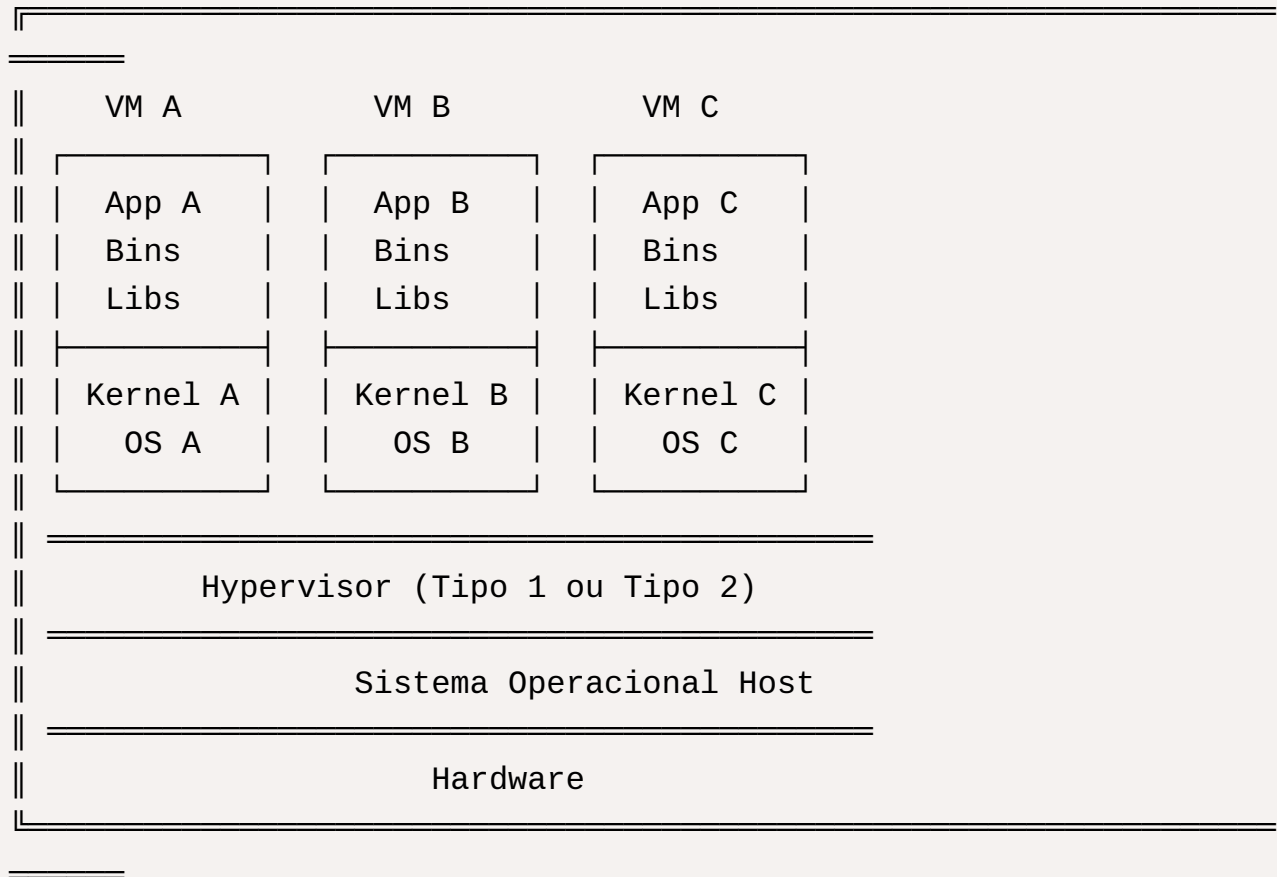
- Limita recursos (CPU, memória)
- Priorização
- Contabilização
- Controle

### 3. Union Filesystem

- Camadas sobrepostas
- Copy-on-write

- Compartilhamento eficiente

## Virtual Machines em Detalhes



## Tipos de Hypervisor

### 1. Tipo 1 (Bare Metal)

- VMware ESXi
- Microsoft Hyper-V
- Citrix XenServer

### 2. Tipo 2 (Hosted)

- VirtualBox
- VMware Workstation

- Parallels

## Análise Técnica Detalhada

### 1. Gerenciamento de Recursos

#### Containers

Recurso	Comportamento
CPU	Compartilhado + Limites cgroups
Memória	Isolada + Limites cgroups
I/O	Compartilhado + Throttling
Network	Virtualizada via namespaces

#### VMs

Recurso	Comportamento
CPU	Dedicado ou compartilhado
Memória	Totalmente isolada
I/O	Emulado ou para-virtualizado
Network	Totalmente virtualizada

### 2. Segurança e Isolamento

#### Containers

Aspecto	Nível
Processo	Alto
Sistema de Arquivos	Alto
Network	Médio
Kernel	Baixo



Hardware	Baixo
Vulnerabilidades	Kernel compartilhado
Escape	Possível

## VMs

Aspecto	Nível
Processo	Total
Sistema de Arquivos	Total
Network	Total
Kernel	Total
Hardware	Alto
Vulnerabilidades	Isoladas por VM
Escape	Muito difícil

## 3. Performance Detalhada

Métrica	Containers	VMs
Boot Time	1-3 seg	30-60 seg
Tamanho Base	~100MB	~1-20GB
Overhead CPU	~1-5%	~10-30%
Overhead Memória	~5-10%	~20-40%
I/O Performance	Nativo	~90-95%
Network Performance	Nativo	~90-95%
Densidade/Host	50-100+	5-10

## Casos de Uso Detalhados

Containers São Ideais Para

## **1. Microserviços**

- Deployments independentes
- Escalabilidade granular
- Ciclo de vida curto

## **2. DevOps**

- CI/CD pipelines
- Testes automatizados
- Ambientes efêmeros

## **3. Cloud Native**

- Kubernetes
- Orquestração
- Auto-scaling

## **4. Desenvolvimento**

- Ambientes consistentes
- Rápida iteração
- Baixo overhead

## **VMs São Ideais Para**

### **1. Infraestrutura Legacy**

- Sistemas monolíticos
- Dependências de SO
- Licenciamento específico

### **2. Isolamento Total**

- Workloads multi-tenant

- Compliance
- Segurança crítica

### 3. Recursos Dedicados

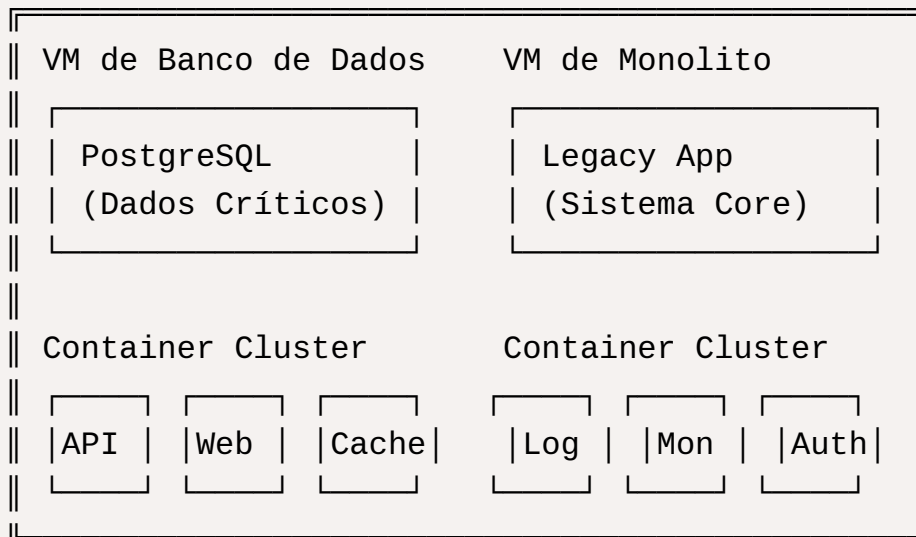
- Garantias de performance
- SLAs estritos
- Cargas previsíveis

### 4. Disaster Recovery

- Snapshots completos
- Backup do estado total
- Migração live

## Cenários Híbridos 🤝

### Exemplo: Arquitetura Moderna



## Melhores Práticas 📖

### Para Containers

1. Uma aplicação por container
2. Imagens imutáveis
3. Use multi-stage builds
4. Minimize camadas
5. Configure health checks
6. Não armazene dados no container
7. Use secrets management

## **Para VMs**

1. Tamanho apropriado
2. Templating e automação
3. Patches regulares
4. Backup estratégico
5. Monitoramento robusto
6. Segurança em camadas
7. Planejamento de capacidade

## **Ferramentas do Ecossistema**

### **Containers**

- Docker
- Podman
- containerd
- LXC/LXD
- rkt

## VMs

- VMware
- Hyper-V
- KVM
- Xen
- VirtualBox

## Waifu Tips 💡

⚠️ **Container-chan diz:** "Lembre-se: containers são efêmeros! Trate-os como gado, não como animais de estimação!"

⚠️ **VM-sama diz:** "VMs são como fortalezas digitais - mais pesadas, mas super seguras!"

## Exercícios Práticos 🏋️

### Nível 1: Básico

1. Compare o tempo de boot entre um container nginx e uma VM com nginx
2. Meça o uso de memória em ambos os casos
3. Teste o isolamento de rede em cada tecnologia

### Nível 2: Intermediário

1. Configure um ambiente híbrido com DB em VM e app em container
2. Implemente limits/requests em containers
3. Configure resource pools em VMs

## Nível 3: Avançado

1. Implemente uma estratégia de backup para ambos
2. Configure monitoring e logging
3. Teste cenários de failover

## Referências

1. Docker Documentation
2. VMware Knowledge Base
3. Linux Container Documentation
4. Kubernetes Documentation
5. Cloud Native Foundation



"A escolha entre containers e VMs é como escolher entre uma katana e um escudo. Às vezes você precisa de velocidade e precisão, outras vezes de proteção máxima."

- Tech Samurai, 2025

# História do Docker

## TIMELINE DO DOCKER

Uma jornada através do tempo e do espaço (de usuário)

## Pré-História dos Containers

### 1979: Unix V7

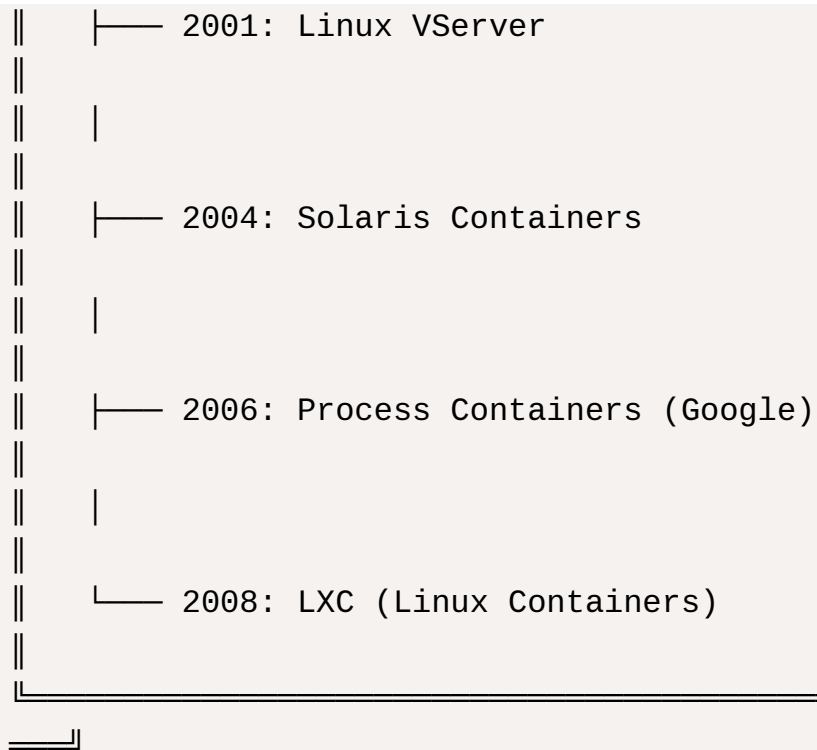
- Introdução do `chroot`
- Primeiro passo para isolamento de processos
- Limitava visão do sistema de arquivos

### Anos 90: Primeiros Passos

- FreeBSD Jails
- Isolamento mais robusto
- Separação de recursos

### 2000-2010: A Era das Fundações

2000 — Virtualização mainstream

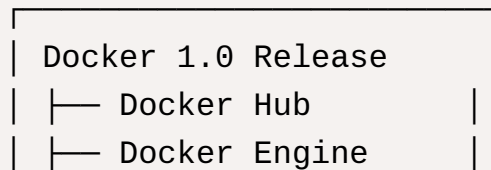


## O Nascimento do Docker

### 2013: O Ano que Mudou Tudo

- Docker é criado por Solomon Hykes
- Parte do projeto dotCloud
- Código aberto desde o início
- Baseado em LXC inicialmente

### 2014: O Ano do Crescimento





| — Comercialização |

## Era Moderna (2015-Presente) 🌟

### 2015: O Ecossistema Explode

- Docker Compose
- Docker Swarm
- Docker Machine
- Docker Toolbox

### 2016: A Maturidade

```
|| Docker for Mac/Windows ||  
|| Containerd              ||  
|| Docker Store            ||
```

### 2017-2019: Consolidação

- Suporte a Kubernetes
- Docker Enterprise
- Multi-stage builds
- BuildKit

### 2020+: Nova Era

- Mirantis acquire Docker Enterprise
- Foco em desenvolvedores

- Docker Desktop subscription
- Inovações contínuas

## Tecnologias Predecessoras

### Virtualização Tradicional

- VMware (1998)
- Xen (2003)
- KVM (2007)

### Containers Primitivos

	chroot (1979)	
	BSD Jails (2000)	
	Solaris Zones	
	Linux VServer	
	OpenVZ	

## Linha do Tempo Detalhada

1979 — chroot  
|  
2000 — FreeBSD Jails  
|  
2001 — Linux VServer  
|  
2004 — Solaris Containers  
|  
2006 — Process Containers (cgroups)  
|  
2008 — LXC

|  
2013 — Docker inicial  
|  
2014 — Docker 1.0  
|  
2015 — Docker Compose/Swarm  
|  
2016 — Docker for Mac/Windows  
|  
2017 — Kubernetes suporte  
|  
2020 — Nova era Docker

## Contribuições Importantes

### Pessoas Chave

- Solomon Hykes (Fundador)
- Andrea Luzzardi (Swarm)
- Michael Crosby (containerd)
- Jessie Frazelle (Security)

### Empresas Pioneiras

1. dotCloud
2. Docker Inc
3. Red Hat
4. Google
5. Microsoft

## Waifu Tips



**History-chan diz:** "Containers são como ninjas modernos - evoluíram de simples ferramentas para mestres do DevOps!"

## Legado e Impacto

### Na Indústria

- Revolução no deployment
- DevOps transformation
- Cloud-native movement
- Microservices adoption

### No Desenvolvimento

- Local development
- CI/CD pipelines
- Testing
- Distribution

## Checkpoint

Você agora sabe:

- [x] Origens do Docker
- [x] Evolução dos containers
- [x] Marcos importantes
- [x] Impacto na indústria

## Exercícios

1. Pesquise sobre uma tecnologia predecessor do Docker
2. Compare containers modernos com Jails do FreeBSD
3. Crie uma linha do tempo visual da evolução dos containers





"Aqueles que não conhecem a história dos containers estão destinados a recriá-los usando Bash scripts."

- DevOps Philosopher, 2025

# Instalação do Docker

## INSTALLATION MATRIX

Escolha sua pílula:

-  Docker Desktop (modo easy)
-  CLI (modo hardcore)

## Requisitos de Sistema

Windows  
Pro/Enterprise/Education

- Windows 10/11
- WSL 2 habilitado
- Virtualização ativada na BIOS

Linux

- Kernel 3.10+
- systemd
- 64-bit

MacOS

- macOS 11+
- Apple Silicon ou Intel
- 4GB RAM (mínimo)

# Docker Desktop vs CLI: Escolha sua Arma 🚬

BATTLE ROYALE: GUI vs CLI

[DOCKER DESKTOP]

VS

[DOCKER CLI]

GUI bonita

Modo hardcore

Métricas fancy

Terminal é vida

Botões clicáveis

Commands only

Newbie friendly

Chad energy

## Docker Desktop: Para os Amantes de GUI 🖱️

### Prós

- Interface bonitinha para seus olhos sensíveis
- Gráficos de uso de recursos (oooh, cores!)
- Kubernetes integrado (porque você *totalmente* vai usar isso)
- Gestão visual de containers (arrasta e solta FTW)

- Extensions marketplace (mais bloat FTW!)

## Contras

- Come RAM como se não houvesse amanhã
- Mais pesado que suas responsabilidades
- Updates constantes (sempre quando você tem deadline)
- Pode viciar em cliques

```
| Resource Usage |  
| RAM: 🔥🔥🔥🔥 |  
| CPU: 🔥🔥🔥 |  
| Disk: 🔥🔥 |  
| Your Soul: Gone |
```

## Docker CLI: Para os Verdadeiros Hackers 🖥️

### Prós

- Leve como uma pluma
- Rápido como Flash
- Scripts automation friendly
- Street cred na comunidade dev
- Funciona até no seu Nokia 3310

### Contras

- Curva de aprendizado vertical
- Man pages são seu novo melhor amigo



- Typos são seu novo pior inimigo
- Precisa decorar comandos (ou usar `history | grep`)

```
$ docker run --help  
> 500 linhas de opções  
> RTFM, n00b!
```

## Qual Escolher? 🤔

### Use Docker Desktop se:

- Você gosta de botões brilhantes
- Tem 32GB+ de RAM sobrando
- Precisa de visualização de métricas
- Está começando no mundo Docker
- Tem medo de terminais pretos

### Use Docker CLI se:

- Você é um verdadeiro cyberpunk
- Seu terminal tem tema neon
- Quer impressionar a crush dev
- Precisa de automação hardcore
- `vim` é seu editor principal

## Waifu Tips 💡

**i** CLI-chan diz: "GUIs são temporárias, terminal é eterno! Mas sério, use o que te faz mais produtivo... baka!"

## Guia de Sobrevivência

### Docker Desktop

```
# Instalação
1. Download installer
2. Next, next, next, finish
3. Rezar para o PC ainda ligar

# Uso
1. Clique nos containers
2. Aperte play/stop
3. Profit!
```

### Docker CLI

```
# Instalação
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
# Agora você é oficialmente 1337

# Comandos básicos
docker ps      # Lista containers (seus pods da Matrix)
docker images  # Lista imagens (seus programas em stand-by)
docker run     # Roda container (hora do show)
docker kill    # Mata container (com vingança)
```

## Checkpoint

Você agora sabe:

- [x] Diferenças entre Desktop e CLI

- [x] Prós e contras de cada um
- [x] Quando usar cada opção
- [x] Como sobreviver com ambos

## Exercícios

1. Instale ambos e veja quanto RAM você perde
2. Tente usar só CLI por uma semana
3. Conte quantas vezes você digitou `docker` errado
4. Faça um alias para seus comandos mais usados
5. Bonus: Customize seu prompt com ASCII art do Docker

## Projetos Práticos

### Projeto 1: "CLI Warrior"

Use apenas CLI por 30 dias. Bônus se seu terminal tiver tema cyberpunk.

### Projeto 2: "GUI Ninja"

Crie um workflow completo usando só Docker Desktop. Sem terminal allowed!



"Na Matrix dos containers, não existe colher... nem mouse." - Morpheus, provavelmente

# Problemas Comuns: Debugando na Matrix

```
DOCKER TROUBLE MATRIX
```

```
[ERROR] - Permission denied
```

```
[ERROR] - Port already in use
```

```
[ERROR] - No space left on device
```

```
[ERROR] - Cannot connect to the Docker daemon
```

## Problemas de Permissão

### Sintoma

```
Got permission denied while trying to connect to the Docker daemon  
socket
```

### Solução

```
# Adicione seu usuário ao grupo docker  
sudo usermod -aG docker $USER
```

```
# Recarregue os grupos (ou faça logout/login)
newgrp docker
```

## Conflito de Portas

### Sintoma

```
Error: Ports are not available: listen tcp 0.0.0.0:3000: bind:
address already in use
```

### Solução

```
# Encontre o processo usando a porta
sudo lsof -i :3000

# Mate o processo (substitua PID)
sudo kill -9 PID

# OU use uma porta diferente
docker run -p 3001:3000 minha-app
```

## Sem Espaço em Disco

### Sintoma

```
no space left on device
```

### Solução

```
# Limpe recursos não utilizados
docker system prune -a --volumes

# Verifique o uso de espaço
docker system df
```

## Daemon Offline

### Sintoma

Cannot connect to the Docker daemon at unix:///var/run/docker.sock

### Solução

```
# Verifique o status do serviço
sudo systemctl status docker

# Reinicie o daemon
sudo systemctl restart docker
```

## Waifu Tips

**i** Debug-chan diz: "Quando tudo falhar, tente `docker system prune`. Mas cuidado, isso é tipo formatar o PC - só que para containers!"

## Problemas de Rede

### Sintoma

ERROR: Network gateway has conflicts with existing routes

### Solução

```
# Recrie a rede default
docker network prune
docker network create bridge
```

## Problemas de Build

### Sintoma

```
Step 3/8 : RUN npm install
npm ERR! code ECONNREFUSED
```

## Solução

```
# Verifique sua conexão
ping 8.8.8.8

# Use mirror alternativo
npm config set registry https://registry.npmmirror.com
```

## Checklist de Debug

### 1. Verifique logs

```
docker logs <container-id>
```

### 2. Inspecione o container

```
docker inspect <container-id>
```

### 3. Entre no container

```
docker exec -it <container-id> sh
```

### 4. Monitore recursos

```
docker stats
```

## Prevenção é o Melhor Remédio

### Boas Práticas

- Use `.dockerignore` apropriadamente

- Mantenha imagens base atualizadas
- Implemente health checks
- Monitore uso de recursos
- Faça backup de volumes importantes

## Ferramentas Úteis

- Portainer (GUI para gestão)
- ctop (top para containers)
- lazydocker (TUI amigável)
- docker-compose (orquestração local)

## Modo Hardcore: Debugging Avançado 🔍

### Análise de Sistema

```
# Verifique limites do sistema
ulimit -a

# Monitore eventos do Docker
docker events

# Debug de rede
docker network inspect bridge
```

### Logs Avançados

```
# Logs em tempo real com timestamp
docker logs -f --timestamps <container-id>

# Últimas 100 linhas
docker logs --tail 100 <container-id>
```



## Recursos de Emergência 🚨

- Docker Debug Guide (<https://docs.docker.com/engine/troubleshoot/>)
- Stack Overflow Docker Tag (<https://stackoverflow.com/questions/tagged/docker>)
- Docker Forum (<https://forums.docker.com/>)



"Todo bug é uma feature que ainda não encontrou seu propósito." - Hackerman

# Containers 101

## CONTAINER BASICS MATRIX

[ Container = Aplicação + Dependências + Runtime ]

"Tudo que você precisa, nada que você não precisa"

## O que é um Container?

Um container é como uma cápsula auto-suficiente que contém:

- Sua aplicação
- Bibliotecas necessárias
- Configurações
- Runtime environment

Sua Aplicação

Dependências

Runtime

## Anatomia de um Container

### Componentes Principais


1. **Namespace** - Isolamento de processos
2. **Cgroups** - Controle de recursos
3. **Union Filesystem** - Sistema de arquivos em camadas
4. **Container Runtime** - Motor de execução

## Por Que Usar Containers?

### Benefícios

- Isolamento consistente
- Portabilidade garantida
- Deploy simplificado
- Escalabilidade fácil
- Versionamento eficiente

## Waifu Tips

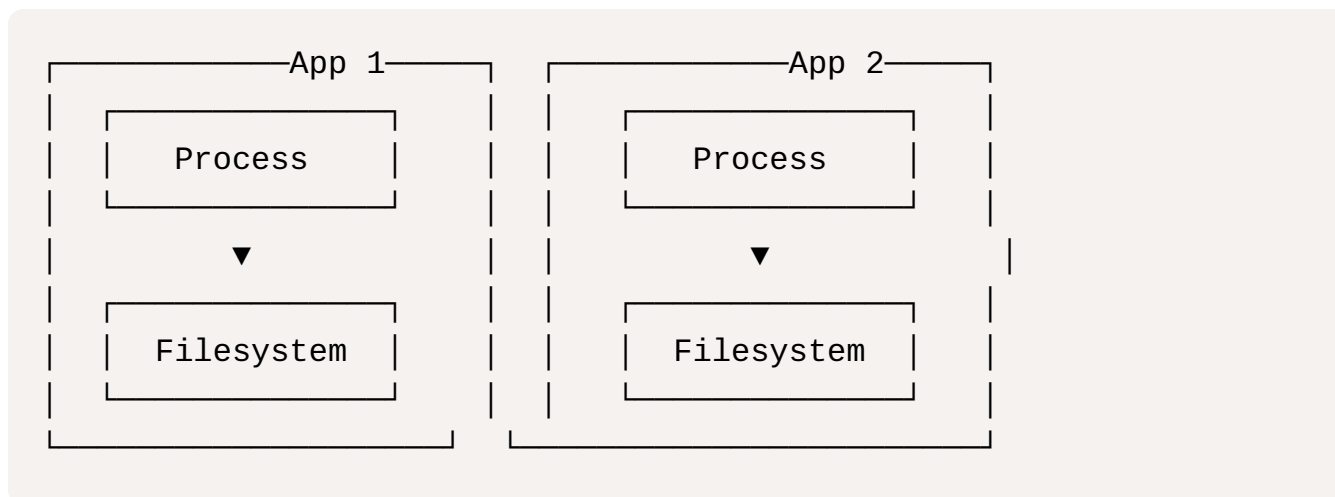
 **Container-chan diz:** "Lembre-se: containers são efêmeros! Se algo importante mudar, será perdido no próximo restart... como lágrimas na chuva!"

## Conceitos Fundamentais

### 1. Imutabilidade

- Containers não devem ser modificados em runtime
- Mudanças através de novas builds
- Tratados como gado, não como pets

## 2. Isolamento



## 3. Recursos Limitados

- CPU controlada
- Memória definida
- I/O gerenciado
- Network isolada

## Primeiros Passos

### Hello World Container

```
# Rode seu primeiro container
docker run hello-world

# Veja o que está rodando
docker ps
```

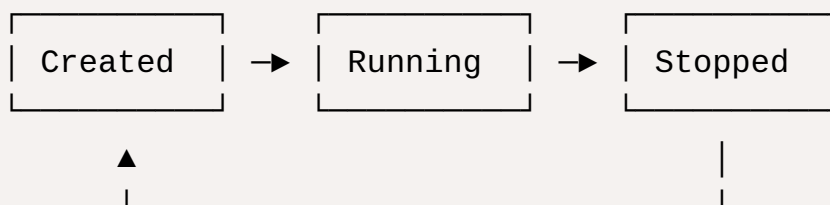
```
# Veja todos os containers
docker ps -a
```

## Container Interativo

```
# Ubuntu container interativo
docker run -it ubuntu bash

# Alpine container leve
docker run -it alpine sh
```

## Container Lifecycle



## Comandos de Ciclo de Vida

```
# Criar container
docker create nginx

# Iniciar container
docker start <container-id>

# Parar container
docker stop <container-id>

# Remover container
docker rm <container-id>
```

## Práticas Recomendadas

## DO's

- Um processo por container
- Use tags específicas
- Defina limites de recursos
- Monitore seus containers
- Use health checks

## DON'Ts

- Não armazene dados no container
- Evite containers muito grandes
- Não use `latest` tag
- Não rode como root
- Não ignore security scans

## Exercícios Práticos

### 1. Hello Container

```
# Rode e entenda o hello-world
docker run hello-world
```

### 2. Container Persistente

```
# Rode nginx com porta exposta
docker run -d -p 8080:80 nginx
```

### 3. Container Exploração

```
# Entre em um container rodando  
docker exec -it <container-id> bash
```

## Debug Básico

### Comandos Essenciais

```
# Logs do container  
docker logs <container-id>  
  
# Processos rodando  
docker top <container-id>  
  
# Estatísticas de uso  
docker stats <container-id>
```

## Checkpoint

Você agora sabe:

- [x] O que são containers
- [x] Como eles funcionam
- [x] Comandos básicos
- [x] Boas práticas
- [x] Ciclo de vida

## Próximos Passos

1. Explore Docker Hub
2. Crie seus próprios containers
3. Aprenda sobre networking

4. Estude volumes
5. Pratique com projetos reais

**⚠** "No mundo dos containers, não existe 'funciona na minha máquina' - porque sua máquina está no container!"

## Deep Dive: Camadas de um Container 🍂

### Union File System

Camada R/W	<- Container Layer
Aplicação	<- Image Layer 3
Dependências	<- Image Layer 2
Sistema Base	<- Image Layer 1

### Como as Camadas Funcionam

1. **Base Layer:** Sistema operacional mínimo
2. **Middleware Layer:** Runtime, libs
3. **Application Layer:** Seu código
4. **Container Layer:** Mudanças em runtime

## Networking Básico 🌐

### Tipos de Rede

```
# Lista redes disponíveis
docker network ls
```



```
# Cria rede customizada
docker network create minha-rede

# Conecta container à rede
docker network connect minha-rede container-id
```

## Modelos de Rede

### 1. Bridge (default)

```
docker run --network bridge nginx
```

### 2. Host (usa rede do host)

```
docker run --network host nginx
```

### 3. None (sem rede)

```
docker run --network none nginx
```

## Storage & Volumes

### Tipos de Storage

#### 1. Volumes (gerenciado pelo Docker)

```
# Cria volume
docker volume create meu-volume

# Usa volume
docker run -v meu-volume:/data nginx
```

#### 2. Bind Mounts (path do host)

```
# Monta diretório local
docker run -v $(pwd):/app nginx
```

### 3. tmpfs (memória)

```
# Armazenamento temporário
docker run --tmpfs /app nginx
```

## Container Security

### Boas Práticas de Segurança

#### 1. Rootless Containers

```
# No Dockerfile
USER non-root-user
```

#### 2. Scan de Vulnerabilidades

```
# Scan de imagem
docker scan nginx:latest
```

#### 3. Limites de Recursos

```
# Limita CPU e memória
docker run --cpus=.5 --memory=512m nginx
```

## Advanced Commands

### Container Stats

```
# Stats em tempo real
docker stats --format "table
{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

```
# Info detalhada
docker inspect container-id
```

## Batch Operations

```
# Para todos containers
docker stop $(docker ps -q)

# Remove containers parados
docker container prune

# Remove tudo
docker system prune -a
```

## Troubleshooting Matrix

### Common Issues

Problema	Comando de Debug	Solução
Container não inicia	<code>docker logs</code>	Verifique logs
Porta em uso	<code>docker port</code>	Mude a porta
Sem espaço	<code>docker system df</code>	Limpe recursos

### Debug Commands

```
# Histórico de comandos
docker history imagem:tag

# Eventos em tempo real
docker events
```

```
# Diff de alterações
docker diff container-id
```

## Container Development Workflow

### Local Development

#### 1. Build Local

```
docker build -t app:dev .
```

#### 2. Hot Reload

```
docker run -v $(pwd):/app app:dev
```

#### 3. Debug Mode

```
docker run --rm -it app:dev sh
```

### Multi-container Setup

```
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: secret
```

### Performance Tuning

## Otimizações

### 1. Multi-stage Builds

```
FROM node:alpine AS builder
WORKDIR /app
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

### 2. Cache Optimization

```
COPY package*.json ./
RUN npm install
COPY . .
```

### 3. Resource Limits

```
docker run \
  --cpus=2 \
  --memory=2g \
  --memory-reservation=1g \
  --memory-swap=4g \
  nginx
```

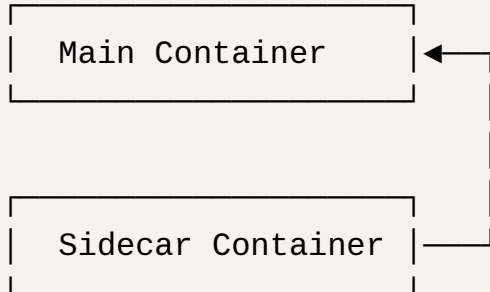
## Waifu Advanced Tips 🎮

**⚠️ Container-chan Advanced Tip:** "Multi-stage builds são como fazer cosplay - você usa diferentes outfits (stages) para chegar no resultado final perfeito!"

**⚠️ Security-chan Warning:** "Nunca confie em imagens de fontes desconhecidas - é como aceitar doces de estranhos!"

# Container Patterns

## Sidecar Pattern



## Ambassador Pattern



## Exercícios Avançados ♂

### 1. Multi-container App

- Crie uma aplicação web com frontend, backend e database
- Use docker-compose
- Implemente volumes persistentes

### 2. Custom Network

- Configure rede bridge customizada
- Conecte múltiplos containers
- Teste comunicação entre containers

### 3. Resource Monitoring

- Configure limites de recursos

- Monitore uso com docker stats
- Implemente health checks

## Certificação Checkpoint

Prepare-se para certificações Docker com estes conceitos:

- [x] Container internals
- [x] Networking modes
- [x] Storage options
- [x] Security best practices
- [x] Resource management
- [x] Troubleshooting
- [x] Container patterns

## Links Úteis

- Docker Hub (<https://hub.docker.com>)
- Docker Docs (<https://docs.docker.com>)
- Play with Docker (<https://labs.play-with-docker.com>)

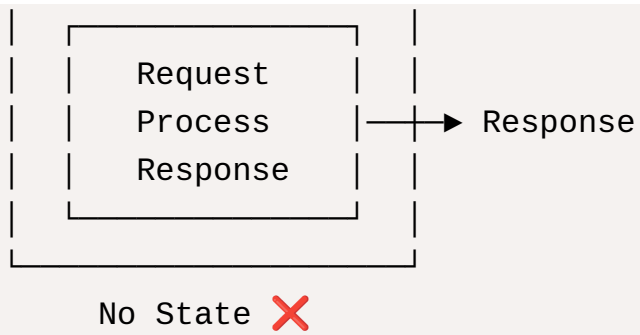


"Um container bem configurado é como um bom código - ele faz exatamente o que deveria fazer, nada mais, nada menos."

## Stateful vs Stateless

### Stateless Containers

Stateless App



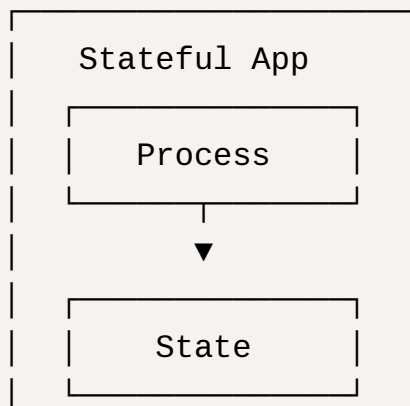
## Características

- Não mantém estado entre requisições
- Facilmente escalável
- Pode ser substituído a qualquer momento
- Ideal para microserviços


## Exemplos de Uso

- Web servers
- API endpoints
- Processamento de filas
- Transformação de dados

## Stateful Containers





Persistent 

## Características

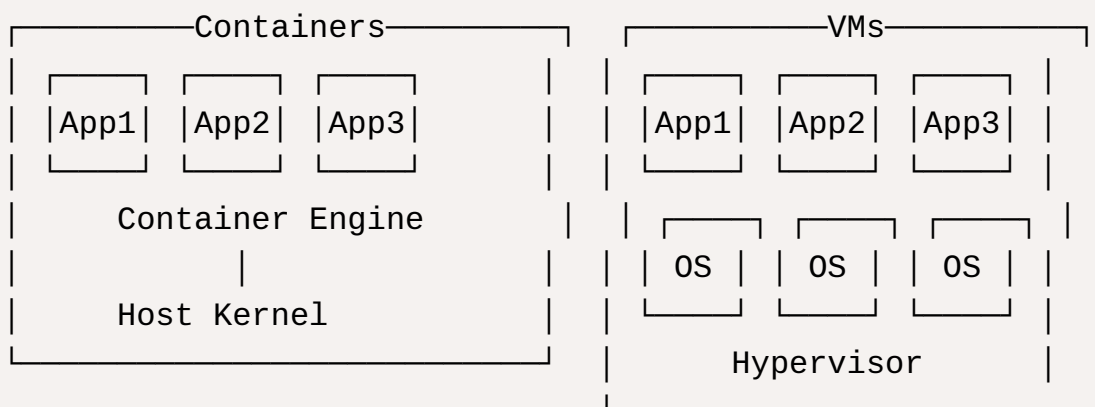
- Mantém dados entre restarts
- Requer volumes persistentes
- Mais complexo para escalar
- Precisa de backup strategy

## Exemplos de Uso

- Databases
- Cache systems
- Message queues
- Session stores

## Containers vs VMs: Análise Detalhada

### Comparação de Arquitetura



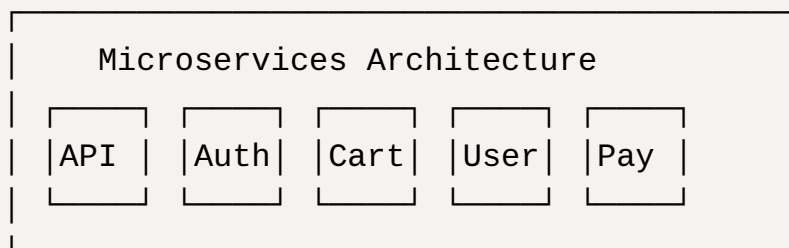
### Matriz de Comparação

Aspecto	Containers	VMs
Boot Time	Segundos	Minutos
Tamanho	MBs	GBs
Performance	Próximo ao bare metal	Overhead significativo
Isolamento	Processo-level	Hardware-level
Portabilidade	Muito alta	Limitada
Segurança	Compartilha kernel	Totalmente isolada
Densidade	Alta (100s)	Baixa (10s)

## Use Cases Detalhados

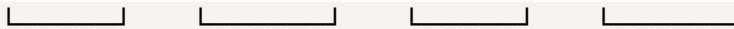
### Ideal para Containers

#### 1. Microserviços

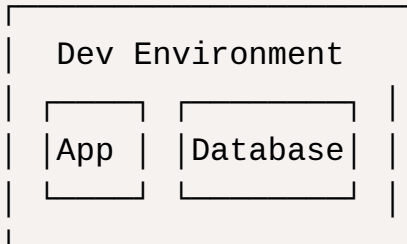


#### 2. CI/CD Pipelines



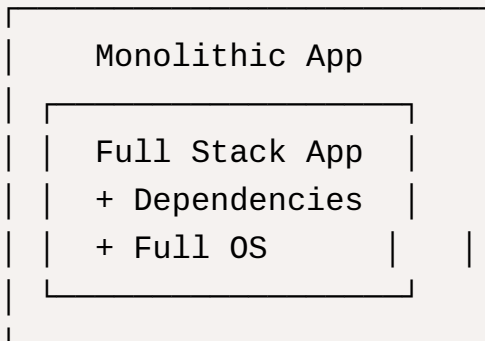


### 3. Desenvolvimento Local

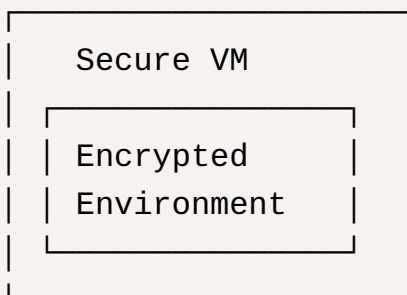


### Ideal para VMs

#### 1. Legacy Applications

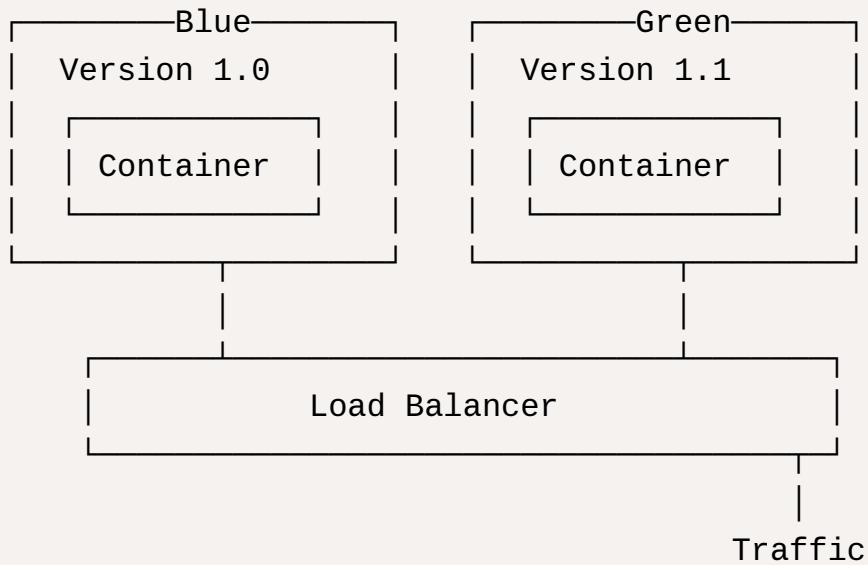


#### 2. Isolamento Completo

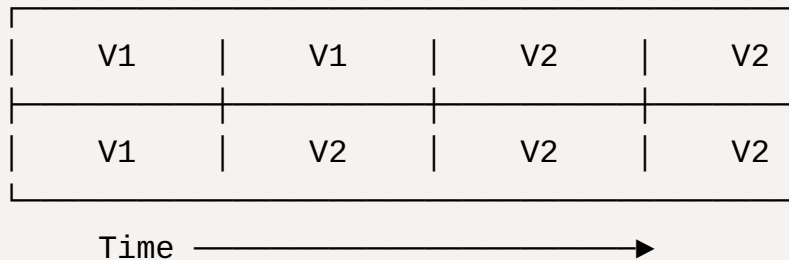


# Padrões de Deployment 🚀

## Blue-Green Deployment

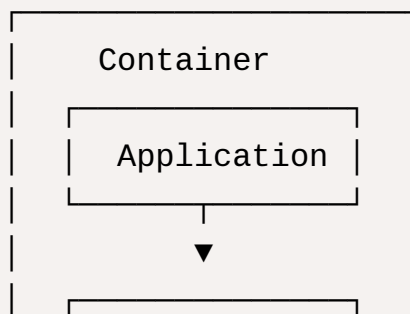


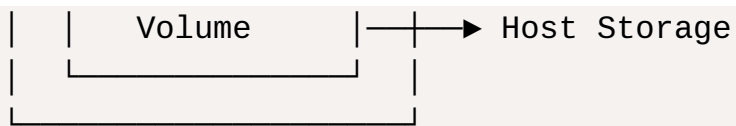
## Rolling Updates



## Estratégias de Persistência 💾

### Volume Types





## 1. Named Volumes

```
docker volume create mydata
docker run -v mydata:/data nginx
```

## 2. Bind Mounts

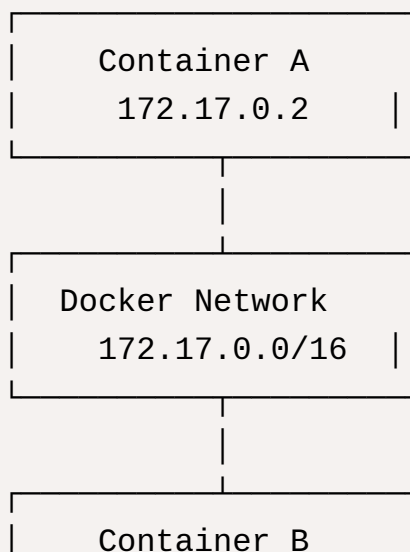
```
docker run -v /host/path:/container/path nginx
```

## 3. tmpfs Mounts

```
docker run --tmpfs /tmp nginx
```

# Networking Avançado 🌐

## Network Types



172.17.0.3

## Network Modes

### 1. Bridge Network

- Default network driver
- Isolated network
- Port mapping

### 2. Host Network

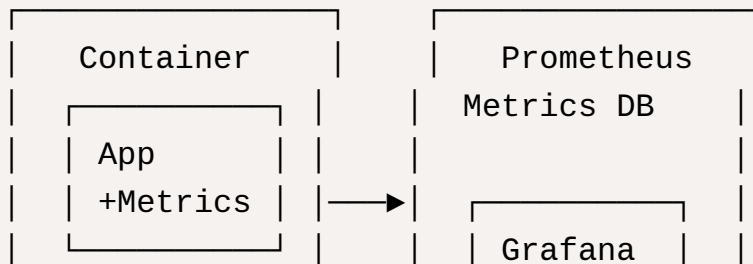
- Uses host networking
- Better performance
- Less isolation

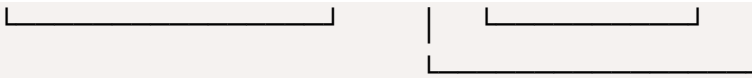
### 3. Overlay Network

- Multi-host networking
- Swarm mode
- Service discovery

## Monitoramento e Logging

### Monitoring Stack





## Logging Patterns

### 1. Stdout/Stderr

```
docker logs container_id
```

### 2. Sidecar Logger

```
services:
  app:
    image: myapp
  logger:
    image: fluentd
  volumes:
    - /var/log:/logs
```

## Waifu Tips 🎮

**⚠️ State-chan diz:** "Stateless é como um café instantâneo - rápido, prático e sempre igual. Stateful é como um café gourmet - precisa de mais cuidado, mas vale a pena!"

**📍 VM-sama avisa:** "VMs são como apartamentos completos, containers são como quartos de hotel - escolha baseado no que precisa!"

## Próximos Passos 🎯

### 1. Explore Docker Hub

2. Crie seus próprios containers
3. Aprenda sobre networking
4. Estude volumes
5. Pratique com projetos reais



"No mundo dos containers, não existe 'funciona na minha máquina' - porque sua máquina está no container!"



# Container Internals: Como Funciona Por Dentro

CONTAINER INTERNALS MATRIX

[ Namespaces + Cgroups + UnionFS = Container ]

"Isolamento, Controle e Eficiência"

## Linux Namespaces

### Tipos de Namespaces Detalhados

#### 1. PID Namespace

Host PID Namespace

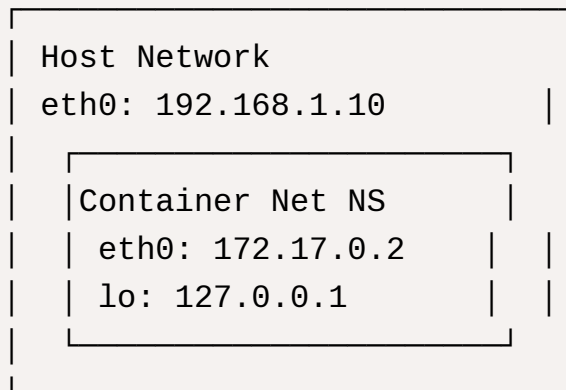
pid: 1234

Container PID NS

pid: 1

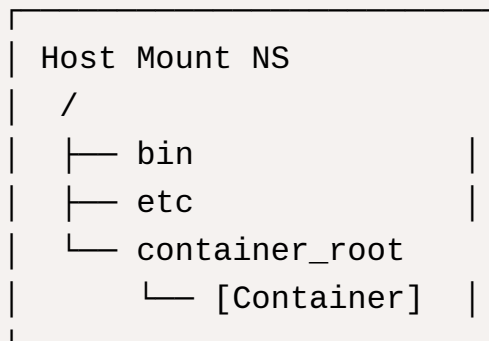
- Isolamento de processos
- Hierarquia própria
- PID 1 como init
- Nested namespaces

## 2. Network Namespace



- Interface virtual
- Routing tables próprias
- Firewall rules
- Socket isolation

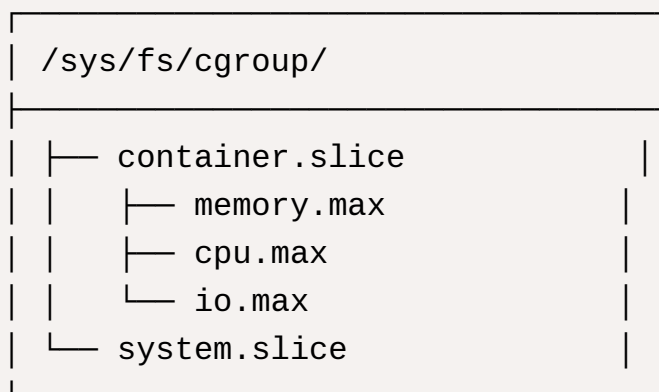
## 3. Mount Namespace



- Filesystem isolation
- Bind mounts
- Pivoting root
- Mount propagation

## Control Groups (cgroups) v2

### Hierarquia Unificada



### Controllers Detalhados

#### CPU Controller

```
# Limitar CPU para 50%
echo "50000 100000" > cpu.max

# Definir peso CPU
echo "100" > cpu.weight
```

#### Memory Controller

```
# Limite de memória
echo "256M" > memory.max
```

```
# Soft limit
echo "200M" > memory.high
```

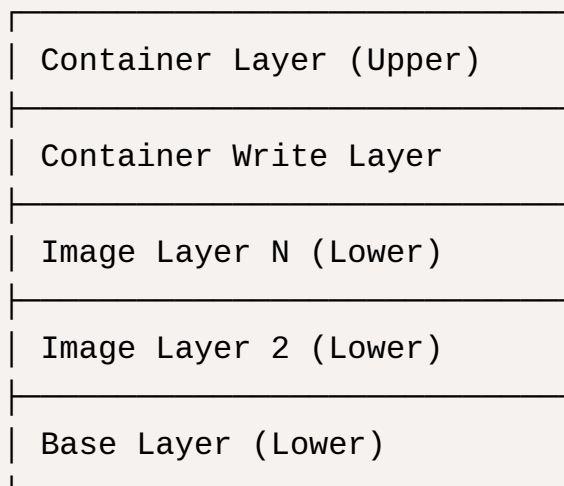
## I/O Controller

```
# Limite de I/O
echo "10485760" > io.max

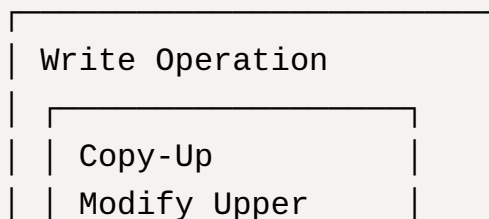
# Peso I/O
echo "100" > io.weight
```

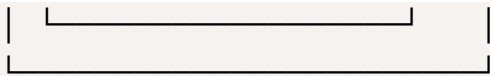
## Union File System Deep Dive

### Overlay2 Internals



### Layer Operations



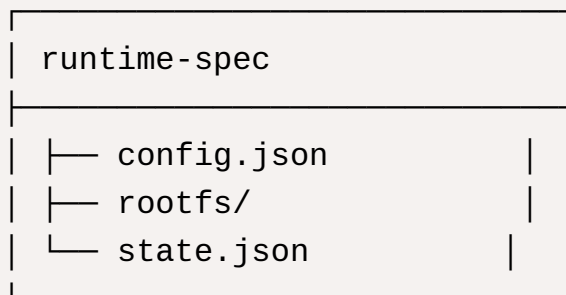


## Directory Structure

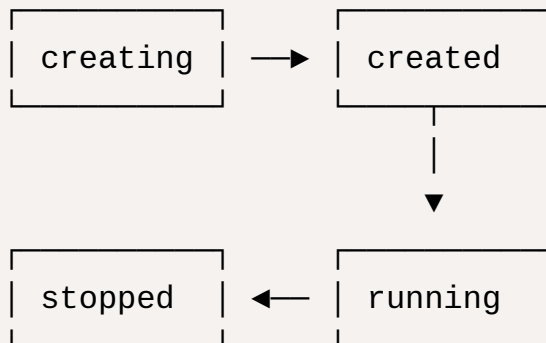
```
/var/lib/docker/overlay2/  
├─ 1/                # Layer shortcuts  
├─ [layer-id]/        # Layer content  
│   ├─ diff/          # Layer fs content  
│   ├─ work/          # Overlay work dir  
│   ├─ merged/        # Mount point  
│   └─ lower          # Lower layer refs
```

## Container Runtime Spec

### OCI Runtime Specification



## Container Lifecycle States



# Security Internals

## Linux Capabilities

Default Container Capabilities	
CHOWN	SETUID
DAC_OVERRIDE	SETGID
FOWNER	NET_BIND
MKNOD	SYS_CHROOT

## Seccomp Profiles

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": ["SCMP_ARCH_X86_64"],
  "syscalls": [
    {
      "names": ["read", "write"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

## AppArmor Profiles

Docker Default Profile	
file,	
network,	
capability,	

```
mount
```

## Advanced Networking

### CNI (Container Network Interface)

```
CNI Architecture
```

```
Plugin
```

```
IPAM
```

### Network Drivers

```
Network Driver Types
```

```
bridge
```

```
host
```

```
overlay
```

```
macvlan
```

```
none
```

## Storage Deep Dive

### Storage Driver Operations

```
Storage Driver Stack
```

```
Container Layer
```

```
Image Layers
```

Storage Driver	
Block Device	

## Volume Types

Volume Types	
Named Volumes	
Bind Mounts	
tmpfs Mounts	

## Advanced Debugging

### Trace Commands

```
# Syscall tracing
strace -p $PID

# Network tracing
tcpdump -i any container_port

# Block I/O tracing
blktrace -d /dev/sda -o trace
```

### Profiling

```
# CPU profiling
perf record -p $PID

# Memory profiling
heaptrack $PID
```



# Performance Tuning ⚡

## Resource Optimization

Performance Factors
CPU Scheduling
Memory Management
I/O Throughput
Network Latency

## Monitoring Metrics

```
# CPU stats
docker stats --format "table {{.Container}}\t{{.CPUPerc}}"

# Memory usage
docker stats --format "table {{.Container}}\t{{.MemUsage}}"
```

## Waifu Tips 🎮

**⚠ Runtime-chan diz:** "OCI é como um contrato mágico que garante que todos os containers falem a mesma língua!"

**⚠ Security-sama avisa:** "Capabilities são como superpoderes - use com responsabilidade!"

**i Network-chan lembra:** "CNI é como um protocolo de comunicação entre dimensões paralelas!"

## Referências Técnicas

### Documentação Oficial

- OCI Runtime Specification
- Linux Kernel Documentation
- Docker Engine Internals
- Container Security Guide

### Ferramentas Úteis

- nsenter
- unshare
- runc
- ctr
- nerdctl

### Debugging Tools

- strace
- tcpdump
- perf
- heaptrack

# Container Runtime: O Motor dos Containers 🚀

CONTAINER RUNTIME MATRIX

[ OCI Runtime + Container Engine + Shim = Container Execution ]

"O maestro que orquestra a execução dos containers"

## Arquitetura do Container Runtime 🏗️

### Camadas de Runtime

High-Level Runtime (Docker)

Container Engine (containerd)

Container Runtime (runc)

Linux Kernel

## Componentes do Kernel

### 1. Namespaces

```
# Listar namespaces
lsns

# Criar novo namespace
unshare --mount --uts --ipc --net --pid --fork --user /bin/bash
```

### 2. Cgroups v2

```
# Criar cgroup
mkdir /sys/fs/cgroup/container1

# Configurar limites
echo "100000" > /sys/fs/cgroup/container1/cpu.max
echo "256M" > /sys/fs/cgroup/container1/memory.max
```

### 3. Capabilities

```
# Listar capabilities
capsh --print

# Dropar capabilities
capsh --drop=cap_sys_admin --
```

## OCI (Open Container Initiative)

### Runtime Specification Detalhada

```
{
  "ociVersion": "1.0.2",
  "root": {
    "path": "rootfs",
    "readonly": true
  },
}
```

```

"process": {
  "terminal": true,
  "user": {
    "uid": 0,
    "gid": 0
  },
  "args": [
    "sh"
  ],
  "env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "TERM=xterm"
  ],
  "cwd": "/",
  "capabilities": {
    "bounding": [
      "CAP_AUDIT_WRITE",
      "CAP_KILL",
      "CAP_NET_BIND_SERVICE"
    ]
  },
},
"mounts": [
  {
    "destination": "/proc",
    "type": "proc",
    "source": "proc"
  }
]
}

```

## Image Manifest Detalhado

```

{
  "schemaVersion": 2,

```

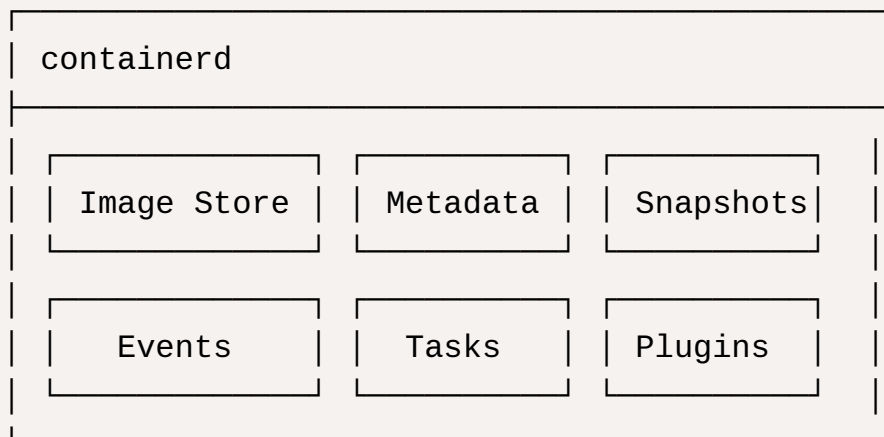
```

"mediaType":
"application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 7023,
    "digest":
"sha256:b5b2b2c507a0944348e0303114d8d93aaaa081732b86451d9bce1f432a
537bc7"
  },
  "layers": [
    {
      "mediaType":
"application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 32654,
      "digest":
"sha256:e692418e4cbaf90ca69d05a66403747baa33ee08806650b51fab815ad7
fc331f"
    }
  ]
}

```

## Container Engine: containerd

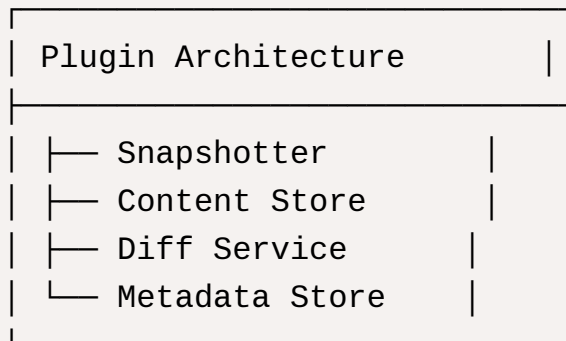
### Arquitetura Detalhada



## API gRPC

```
service Containers {  
    rpc Create(CreateContainerRequest) returns  
    (CreateContainerResponse);  
    rpc Get(GetContainerRequest) returns (GetContainerResponse);  
    rpc List(ListContainersRequest) returns  
    (ListContainersResponse);  
    rpc Delete(DeleteContainerRequest) returns  
    (DeleteContainerResponse);  
    rpc Update(UpdateContainerRequest) returns  
    (UpdateContainerResponse);  
}
```

## Plugins Avançados



## Low-Level Runtime: runc

### Internals do runc

```
// libcontainer/configs/config.go  
type Config struct {  
    Rootfs      string    `json:"rootfs"`  
    ReadonlyFs  bool      `json:"readonly"`  
    Namespaces  []Namespace `json:"namespaces"`  
    Cgroups     *Cgroup    `json:"cgroups"`  
    Devices     []*Device  `json:"devices"`  
    Mounts      []*Mount   `json:"mounts"`  
}
```

```

Hooks      Hooks      `json:"hooks"`
}

```

## Hooks do Lifecycle

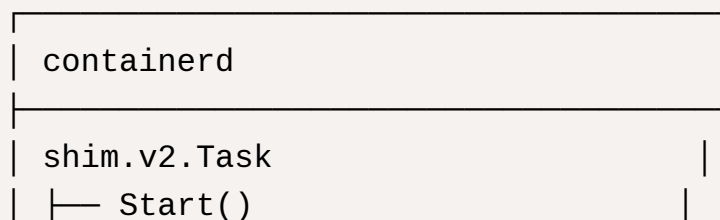
```

{
  "prestart": [
    {
      "path": "/usr/bin/fix-mounts",
      "args": ["fix-mounts", "arg1"],
      "env": [ "key1=value1" ]
    }
  ],
  "poststart": [
    {
      "path": "/usr/bin/notify-start",
      "timeout": 5
    }
  ],
  "poststop": [
    {
      "path": "/usr/bin/cleanup",
      "args": ["cleanup", "arg1"]
    }
  ]
}

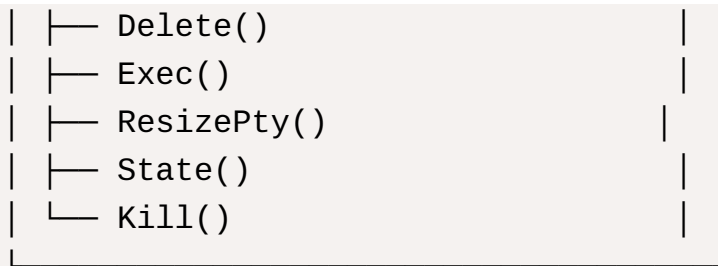
```

## Container Shim v2

### Arquitetura do Shim







## Implementação do Shim

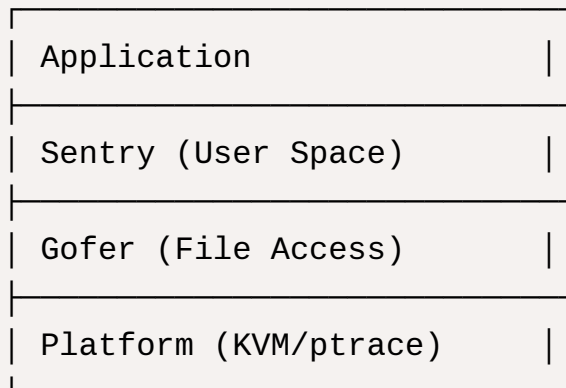
```

type Shim interface {
    StartShim(ctx context.Context, id string, opts ShimOpts)
(string, error)
    Cleanup(ctx context.Context) (*Exit, error)
    Connect(ctx context.Context, onClose func()) error
}

```

## Runtime Alternatives Detalhadas

### gVisor Internals



### Kata Containers Architecture





## Advanced Security Features

### SELinux Policies

```
# Política para container
cat > container.te <<EOF
policy_module(container, 1.0)
require {
    type container_t;
    type container_file_t;
    class file { read write };
}
allow container_t container_file_t:file { read write };
EOF
```

### AppArmor Profile Detalhado

```
profile docker-default flags=(attach_disconnected,mediate_deleted)
{
    network,
    capability,
    file,
    umount,

    deny @{PROC}/{*,**^[0-9*],sys/kernel/shm*} wx,
    deny @{PROC}/sysrq-trigger rwklx,
    deny @{PROC}/mem rwklx,
    deny @{PROC}/kmem rwklx,
}
```

# Advanced Networking Features

## CNI Plugin Implementation

```
type CNI interface {  
    Setup(ctx context.Context, id string, path string, opts  
    ...Opt) (*CNIResult, error)  
    Remove(ctx context.Context, id string, path string, opts  
    ...Opt) error  
    Load(opts ...Opt) error  
    Status() error  
}
```

## Network Namespace Configuration

```
# Criar veth pair  
ip link add veth0 type veth peer name veth1  
  
# Mover para namespace  
ip link set veth1 netns container1  
  
# Configurar IP  
nsenter -t $PID -n ip addr add 172.17.0.2/16 dev eth0
```

## Storage Internals

### Overlay2 Driver



```
| Lower Dir |
```

## Volume Driver Plugin

```
type Driver interface {  
    Create(name string, opts map[string]string) error  
    Remove(name string) error  
    Path(name string) string  
    Mount(name string) (string, error)  
    Unmount(name string) error  
}
```

## Advanced Debugging Techniques 🔍

### Tracing com BPF

```
# Trace syscalls  
bpftrace -e 'tracepoint:syscalls:sys_enter_* /pid == 1234/ {  
    @[probe] = count(); }'  
  
# Trace network  
bpftrace -e 'tracepoint:net:netif_receive_skb { @packets =  
    count(); }'
```

### Core Dumps Analysis

```
# Gerar core dump  
echo "core.%e.%p" > /proc/sys/kernel/core_pattern  
  
# Analisar com gdb  
gdb `which containerd` /tmp/core.containerd.12345
```

## Performance Profiling ⚡

## CPU Profiling

```
# Profiling com perf
perf record -F 99 -p $PID -g -- sleep 30

# Análise
perf report --stdio
```

## Memory Analysis

```
# Heap profile
go tool pprof http://localhost:6060/debug/pprof/heap

# Goroutine analysis
go tool pprof http://localhost:6060/debug/pprof/goroutine
```

## Runtime Metrics & Monitoring

### Prometheus Metrics

```
- job_name: 'containerd'
  static_configs:
    - targets: ['localhost:9100']
  metrics_path: '/metrics'
```

### Custom Metrics

```
var (
    containerStarts =
    prometheus.NewCounter(prometheus.CounterOpts{
        Name: "container_starts_total",
        Help: "Total number of container starts",
    })
)
```

## Advanced Configuration

### Runtime Config


```
[plugins."io.containerd.grpc.v1.cri"]
  stream_server_address = "127.0.0.1"
  stream_server_port = "0"
  enable_selinux = false
  enable_tls_streaming = false


[plugins."io.containerd.grpc.v1.cri".containerd]
  snapshotter = "overlayfs"
  default_runtime_name = "runc"
```


### Custom Runtime Handler

```
runtime_handler_pool:
  runtime_handlers:
    - name: "custom-handler"
      runtime_type: "io.containerd.runc.v2"
      runtime_root: "/run/containerd/custom-handler"
      privileged_without_host_devices: false
```

## Waifu Advanced Tips

 **Kernel-chan sugere:** "Namespaces e cgroups são como os poderes mágicos dos containers - use-os com sabedoria!"

 **Security-sama alerta:** "AppArmor e SELinux são seus guardiões - nunca os desative em produção!"

 **Debug-chan ensina:** "strace e perf são suas ferramentas de investigação - domine-as para resolver qualquer mistério!"

# Referências Avançadas

## Especificações

- OCI Runtime Spec v1.0 (<https://github.com/opencontainers/runtime-spec>)
- Container Network Interface Spec (<https://github.com/containernetworking/cni>)
- Image Spec v1.0 (<https://github.com/opencontainers/image-spec>)

## Documentação Técnica

- containerd Architecture
- runc Internals
- Linux Kernel Namespaces
- Control Groups v2

## Ferramentas de Debug

- runc debug
- ctr debug
- crictl debug
- containerd-debug

## Recursos de Performance

- perf-tools
- bpftrace
- systemtap
- ftrace

## Segurança

- AppArmor Profiles
- SELinux Policies
- Seccomp Filters
- Linux Capabilities

## Networking

- CNI Plugins
- Network Namespaces
- iptables
- tc (traffic control)

## Storage

- Overlay2
- Device Mapper
- BTRFS
- ZFS

## Glossário

### Termos Essenciais

OCI	: Open Container Initiative	
CNI	: Container Network Interface	
Seccomp	: Secure Computing Mode	
AppArmor	: Sistema MAC de segurança	
SELinux	: Security-Enhanced Linux	




cgroups	: Control Groups
---------	------------------

## Componentes do Runtime

containerd	: Container engine de alto nível
runc	: Runtime OCI de baixo nível
shim	: Processo intermediário

# Segurança de Containers

 "Na matrix dos containers, a segurança não é uma opção - é uma arte marcial."  
- Manifesto do Security-Sama

## O Dojo da Segurança

SECURITY MATRIX: DEFCON 1

"Proteja seus containers  
como se o hack dependesse  
disso... porque depende."

## Fundamentos de Segurança

### Princípios Básicos

- Princípio do menor privilégio
- Defense in depth
- Segregação de responsabilidades
- Fail-safe defaults
- Economia de mecanismos

### Modelo de Ameaças

1. Ataques de rede
2. Escalação de privilégios
3. Vulnerabilidades de imagem

- 4. Configurações incorretas
- 5. Malware e código malicioso

## As Cinco Artes Marciais da Segurança

### 1. Namespaces: O Isolamento Perfeito

```
# O ritual do isolamento
docker run --pid=host --ipc=host nginx
# "Como um monge em sua montanha digital"
```

#### Tipos de Namespaces

- PID Namespace
- Network Namespace
- Mount Namespace
- UTS Namespace
- IPC Namespace
- User Namespace

### 2. Capabilities: O Poder Controlado

```
# Dropping powers like it's hot
docker run --cap-drop ALL --cap-add NET_BIND_SERVICE nginx
# "Com grandes poderes vêm grandes vulnerabilidades"
```

#### Capabilities Essenciais

- CAP\_NET\_BIND\_SERVICE
- CAP\_CHOWN
- CAP\_DAC\_OVERRIDE

- CAP\_SETGID
- CAP\_SETUID
- CAP\_SETFCAP

### 3. SecComp: O Filtro Místico

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": ["SCMP_ARCH_X86_64"],
  "syscalls": [
    {
      "names": ["accept", "bind"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

#### Perfis SecComp Comuns

- Default Docker Profile
- Custom Restricted Profile
- No New Privileges Flag

### 4. AppArmor: O Escudo Invisível

```
# Profile de proteção
docker run --security-opt apparmor=docker-default nginx
# "Protegido por forças além da compreensão mortal"
```

#### Políticas AppArmor

- Controle de acesso a arquivos
- Capacidades de rede

- Montagem de sistemas de arquivos
- Execução de programas

## 5. SELinux: O Guardião Ancestral

```
# Modo enforcing ativado
docker run --security-opt label=level:s0:c100,c200 nginx
# "Porque às vezes, paranoia é apenas bom senso"
```

### Contextos SELinux

- Processo
- Arquivo
- Porta
- Usuário

## Segurança em Camadas

### 1. Segurança da Imagem

- Imagens base mínimas
- Multi-stage builds
- Scanning de vulnerabilidades
- Versionamento explícito
- Políticas de atualização

### 2. Segurança do Runtime

- Limites de recursos
- Health checks
- Logging e monitoramento

- Políticas de reinicialização
- Gestão de secrets

### 3. Segurança da Rede

- Redes isoladas
- Firewalls internos
- TLS mútuo
- Service mesh
- Network policies

## Best Practices: O Caminho do Guerreiro

### DO's

- |                             |
|-----------------------------|
| 1. Minimize a superfície    |
| 2. Use imagens oficiais     |
| 3. Scan de vulnerabilidades |
| 4. Updates regulares        |
| 5. Monitore tudo            |

### DON'Ts

- |                                |
|--------------------------------|
| 1. Nunca use root              |
| 2. Não ignore warnings         |
| 3. Evite portas desnecessárias |
| 4. Não guarde secrets no code  |
| 5. Não confie em ninguém       |

# Secrets Management: A Arte do Sigilo 🤫

## Vault-chan's Tips

```
# Gerenciando secrets como um ninja
docker secret create app_secret secret.txt
docker service create --secret app_secret nginx
# "Seus segredos estão seguros comigo, senpai!"
```

## Estratégias de Gestão de Secrets

1. Docker Secrets
2. HashiCorp Vault
3. AWS Secrets Manager
4. Azure Key Vault
5. GCP Secret Manager

# Container Hardening: A Forja Digital 🗡️

## Dockerfile Fortificado

```
# Base segura
FROM alpine:latest AS builder
RUN adduser -D appuser
USER appuser

# Multi-stage para minimizar superfície
FROM scratch
COPY --from=builder /etc/passwd /etc/passwd
USER appuser
```

## Checklist de Hardening

1. Usuário não-root
2. Filesystem read-only
3. Capabilities mínimas
4. Seccomp profile
5. AppArmor/SELinux

## Security Scanning: O Olho que Tudo Vê

### Ferramentas de Scanning

1. Trivy
2. Clair
3. Anchore
4. Snyk
5. Docker Scan

### Trivy: O Scanner Místico

```
# Invocando o scanner
trivy image nginx:latest
# "Deixe que os olhos do void examinem seu código"
```

## Runtime Protection: A Guarda Eterna

### Falco: O Sentinela

```
- rule: Terminal shell in container
  desc: A shell was spawned by a container
  condition: container.id != host and proc.name = bash
  output: Shell spawned in a container (user=%user.name)
```



```
container=%container.name)  
priority: WARNING
```

## Ferramentas de Runtime Protection

1. Falco
2. Aqua Security
3. Twistlock
4. NeuVector
5. StackRox

## Incident Response: O Plano de Batalha 🚨

### O Protocolo do Caos

1. Detectar: "Algo se move nas sombras..."
2. Conter: "Sele as brechas!"
3. Erradicar: "Purificação digital!"
4. Recuperar: "Ressurreição dos sistemas!"

### Playbooks de Resposta

1. Container Compromise
2. Image Vulnerability
3. Network Breach
4. Resource Exhaustion
5. Privilege Escalation

# Compliance e Auditoria


## Frameworks de Compliance


- CIS Docker Benchmark
- NIST Container Security
- PCI DSS
- HIPAA
- SOC 2

## Ferramentas de Auditoria

1. Docker Bench Security
2. InSpec
3. Dockle
4. Lynis
5. OpenSCAP

## Waifu Security Tips

 **Security-chan** avisa: "Containers são como castelos - proteja todas as entradas, ou prepare-se para invasões! uwu"

 **Firewall-sama** declara: "Uma porta aberta é como um convite para o caos digital! Ara ara~"

## Laboratório Prático

### Exercício 1: Configuração Básica

1. Configure um container rootless
2. Implemente AppArmor profile
3. Configure network policies
4. Estabeleça resource limits
5. Implemente health checks

## **Exercício 2: Hardening Avançado**

1. Multi-stage build
2. SecComp profile customizado
3. SELinux policies
4. Docker Secrets
5. Scanning automation

## **Exercício 3: Monitoramento**

1. Configure Falco
2. Implemente logging
3. Setup alerting
4. Trace syscalls
5. Analyze metrics

## **Palavras Finais**

Como diria o lendário Security Master: "Na matrix dos containers, não existem sistemas 100% seguros - apenas níveis diferentes de paranoia."

## **Recursos Adicionais**

**Documentação Oficial**

- Docker Security (<https://docs.docker.com/engine/security/>)
- Kubernetes Security (<https://kubernetes.io/docs/concepts/security/>)
- Container Security Guide  
(<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>)

## Ferramentas

- Docker Bench Security (<https://github.com/docker/docker-bench-security>)
- Trivy (<https://github.com/aquasecurity/trivy>)
- Falco (<https://falco.org/>)

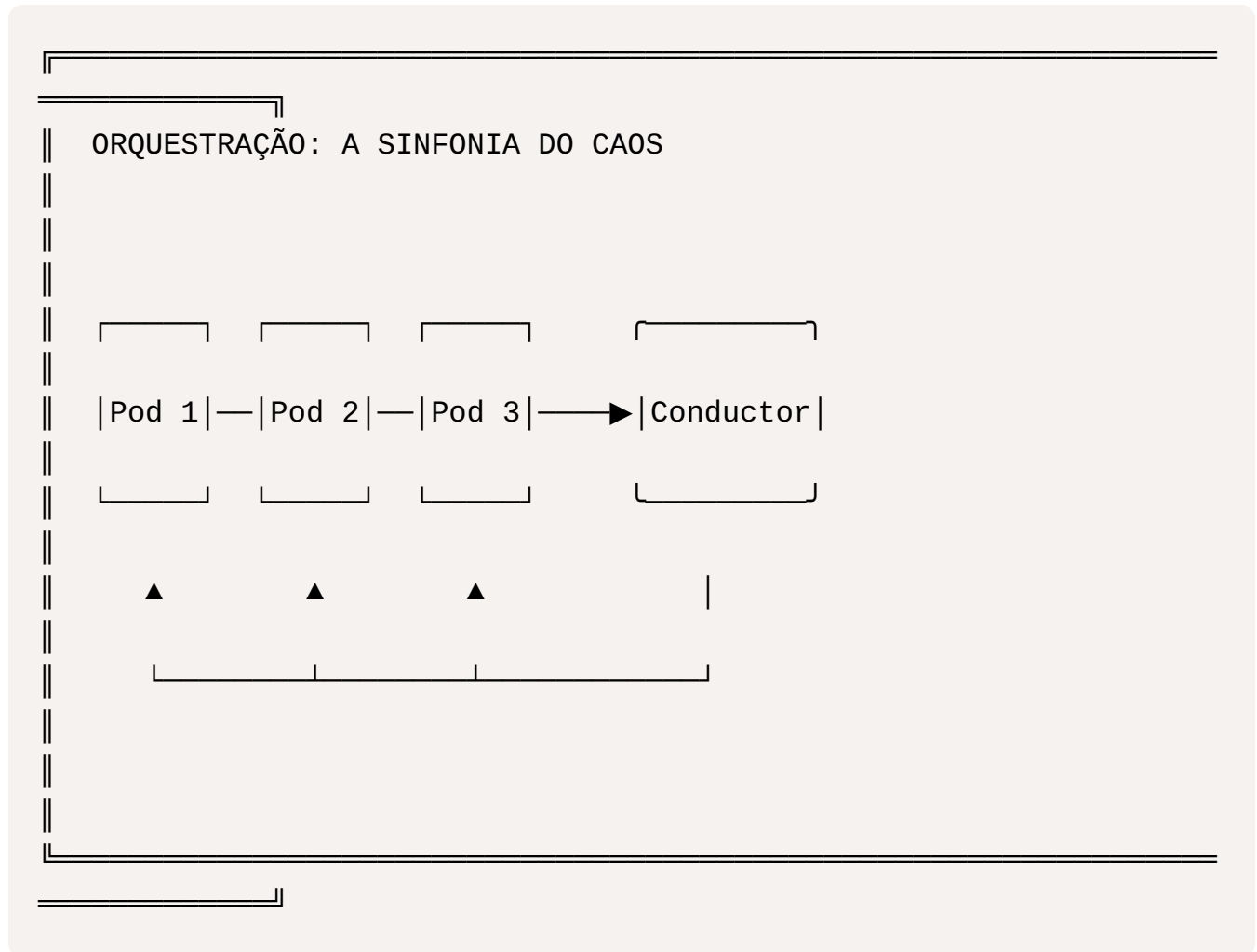
## Comunidade

- Docker Security Mailing List
- Container Security Working Group
- CNCF Security SIG



"E lembre-se: quando a noite digital cai, apenas seus controles de segurança separam seus containers do caos." - Memórias de um Security Engineer, 2084

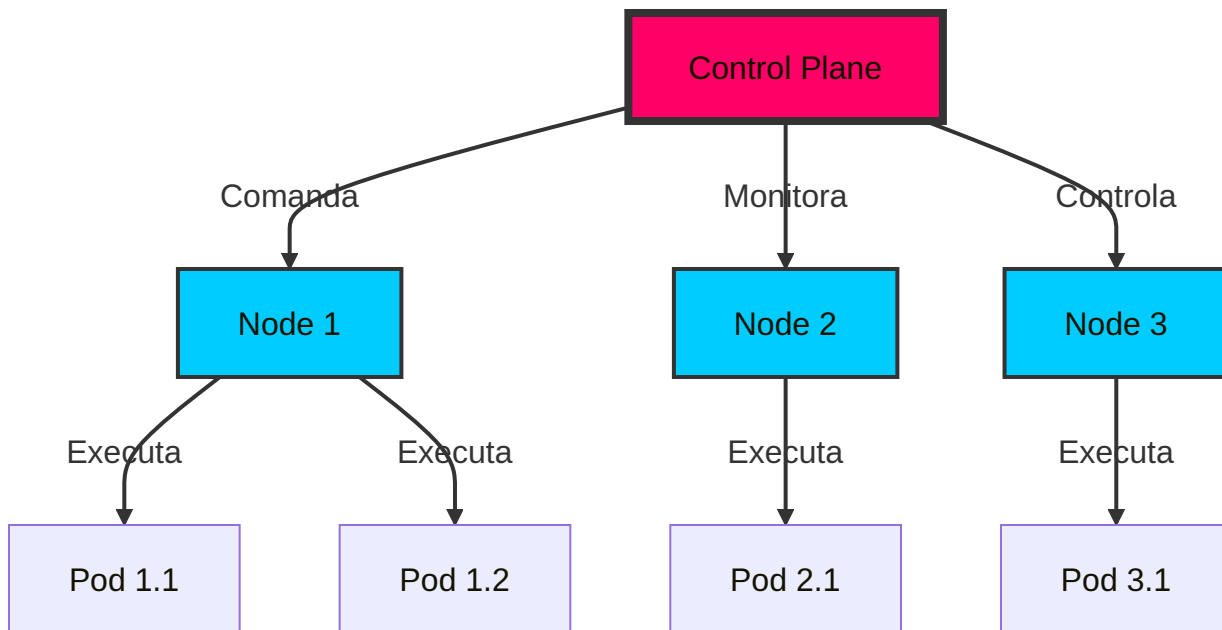
# Orquestração de Containers: A Sinfonia do Caos Digital



Ah, caro leitor, adentre comigo este submundo digital onde containers dançam sua valsa efêmera, nascendo e morrendo como sonhos em uma noite de neón. Como num romance gótico-tech, onde cada microsserviço é um personagem em busca de seu propósito existencial, a orquestração emerge como o narrador onisciente, guiando destinos em meio ao caos dos datacenters.

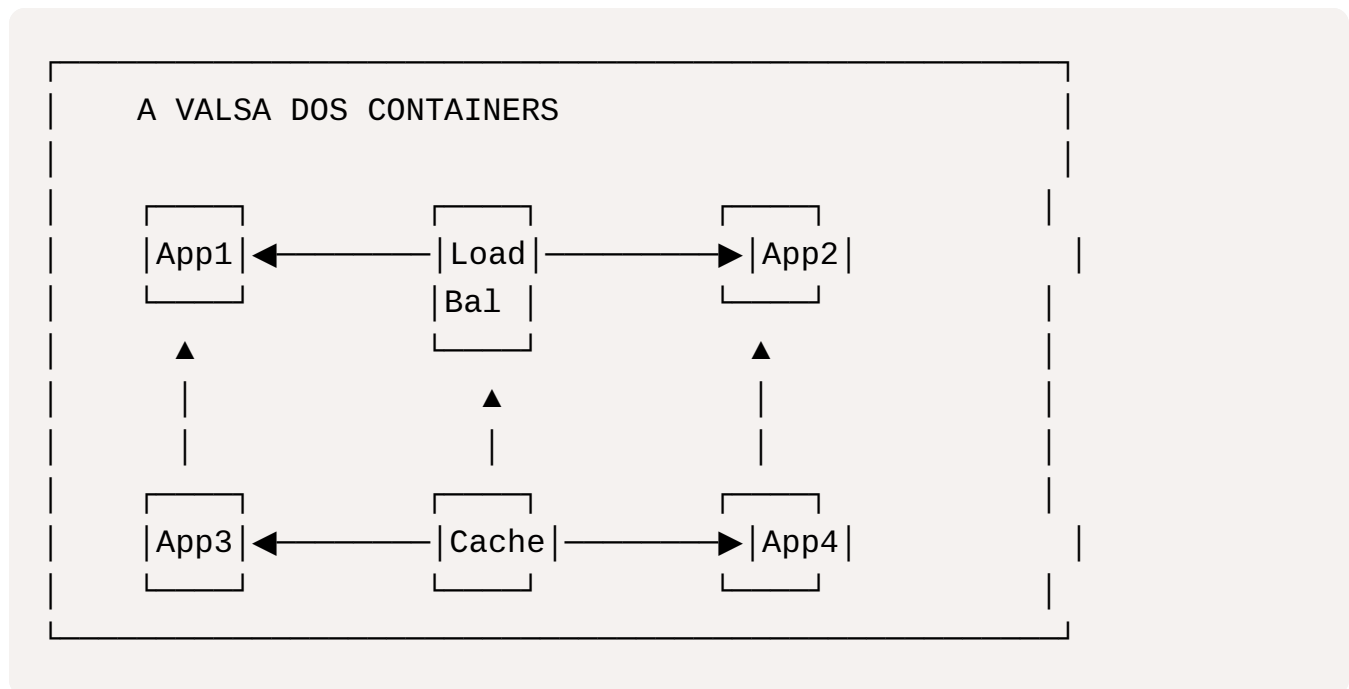
## A Anatomia do Caos Controlado

Imagine, se puder, uma metrópole vertical onde cada container é um apartamento em um arranha-céu infinito. Sem a orquestração, teríamos o caos - containers brigando por recursos como moradores disputando o último pedaço de pizza no elevador.



## A Dança dos Containers

Como um baile de máscaras em uma festa corporativa distópica, cada container executa sua coreografia precisamente ensaiada. O load balancer, qual mestre de cerimônias cibernético, direciona o fluxo de requests como um DJ mixando batidas em uma rave underground.



## Conceitos Fundamentais

## Orquestração

Imagine uma sinfonia caótica, onde cada container é um músico rebelde tocando seu próprio instrumento. A orquestração é nosso maestro cibernético, transformando essa cacofonia digital em uma sinfonia harmônica. Como um controle de tráfego aéreo para seus microsserviços, só que com menos café e mais automação.

Em vez de ter DevOps insones monitorando dashboards intermináveis (embora isso ainda aconteça, não nos enganemos), a orquestração é nosso piloto automático em meio ao caos. É como ter um mordomo digital que:

- Ressuscita containers mortos como um necromante high-tech
- Equilibra carga como um malabarista quantum
- Escala aplicações mais rápido que fofocas em rede social
- Auto-cura como um Wolverine dos microsserviços

## Principais Orquestradores

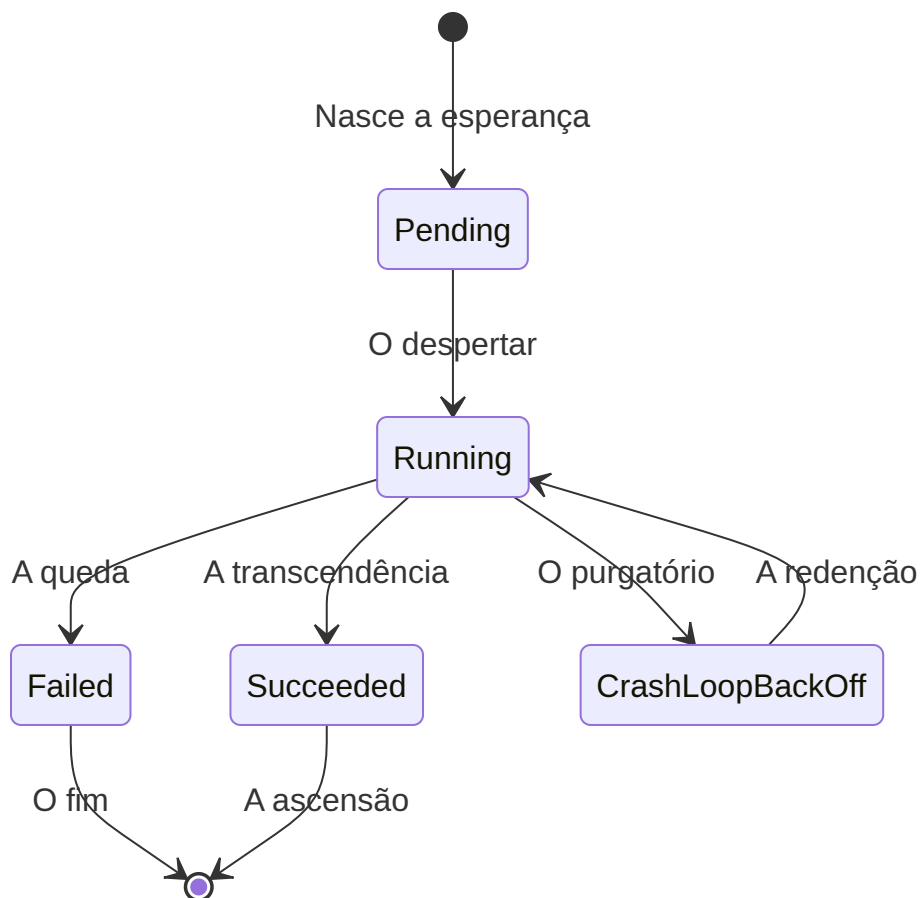
Como num mercado negro de implantes cibernéticos, cada orquestrador tem sua especialidade:

- Kubernetes: O titã cromado, complexo como um romance russo, mas igualmente recompensador
- Docker Swarm: O irmão mais simples, para quando você não quer um doutorado só para fazer deploy
- Apache Mesos: O veterano de guerra, cicatrizes de batalhas corporativas incluídas
- Nomad: O infiltrado HashiCorp, minimalista como um apartamento japonês

## Kubernetes: O Leviatã Digital

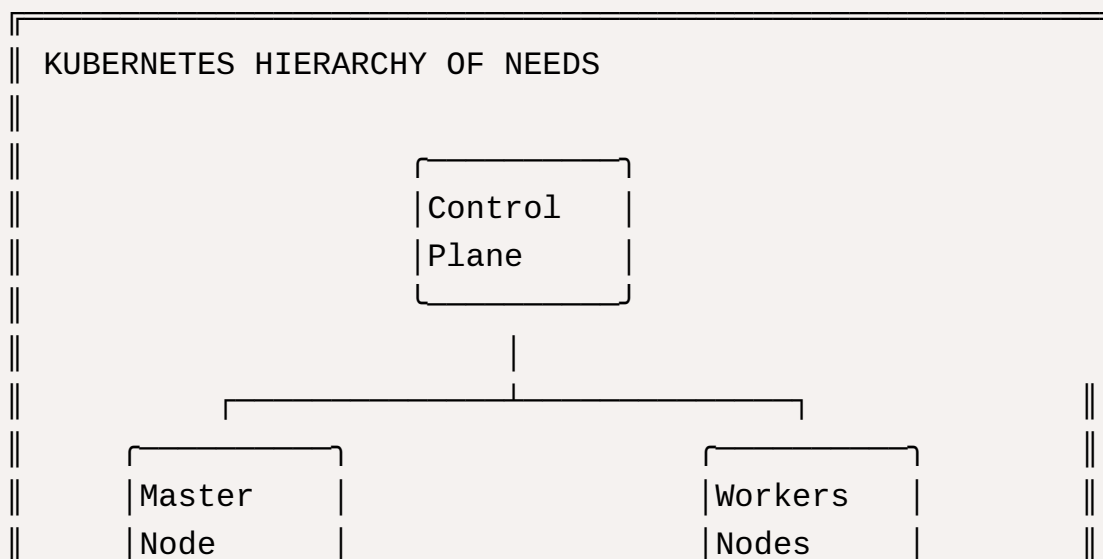
Ah, Kubernetes! O leviatã dos mares digitais, onde cada tentáculo é um controller e cada escama um pod. Como um romance de Lovecraft reescrito em YAML, sua complexidade é tanto seu charme quanto sua maldição.

## O Ciclo de Vida: Uma Tragédia em Cinco Atos

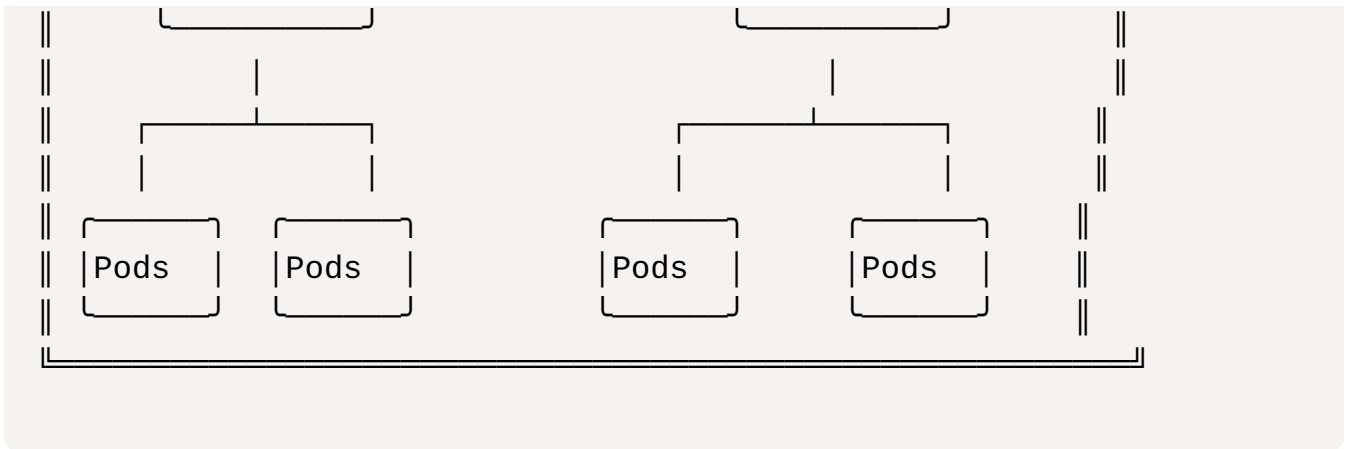


Cada pod é um protagonista em sua própria novela existencial. Alguns vivem vidas longas e prósperas, servindo requests até o sunset de sua última replica. Outros, qual velas ao vento, apagam-se no primeiro sopro de um OOMKilled.

## A Hierarquia do Poder: Uma Distopia Corporativa

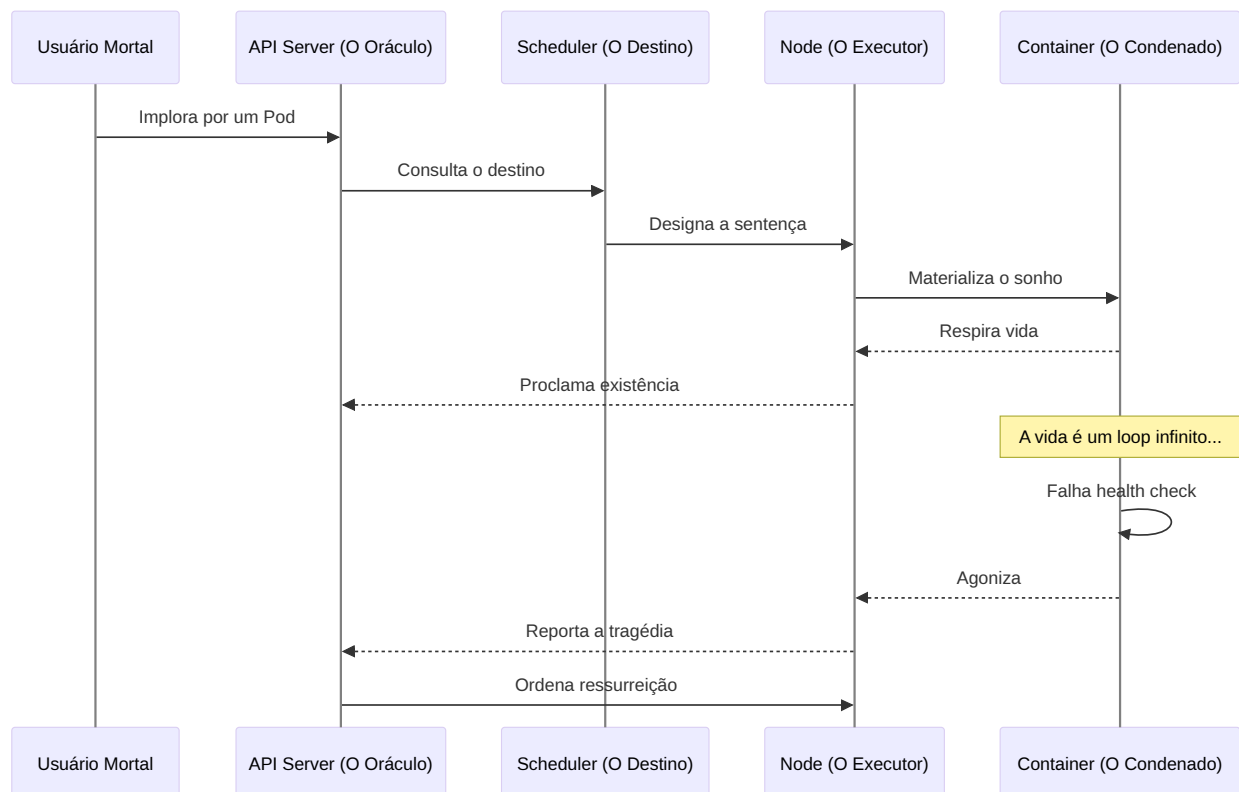






O Control Plane, qual CEO de uma megacorporação, comanda seu império de microsserviços do alto de sua torre de marfim. Os nodes, trabalhadores incansáveis, executam suas ordens sem questionar, enquanto pods nascem e morrem ao sabor dos deployments.

## A Coreografia do Caos



## Componentes Básicos

Como uma corporação megacapitalista distópica, K8s tem sua própria hierarquia:

- Control Plane: O conselho diretor do seu datacenter, tomando decisões que afetam a vida de milhares de containers
- Nodes: Trabalhadores incansáveis, como replicantes em uma linha de montagem digital
- Pods: Pequenos apartamentos compartilhados onde containers vivem em harmonia (ou não)
- Services: O sistema de metrô do seu cluster - conectando pods como estações em uma malha neon
- Deployments: Manifestos digitais que descrevem seu mundo ideal (que nunca existe exatamente como planejado)

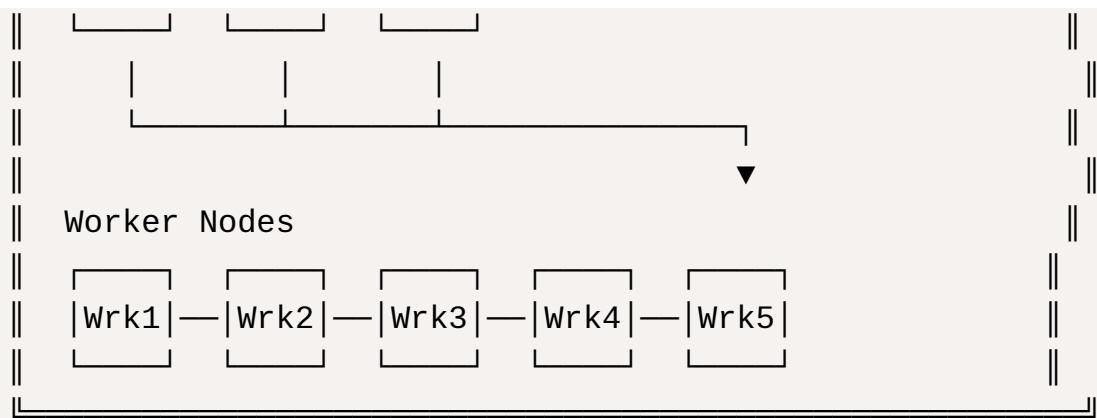
## Arquitetura

- API Server: O cérebro do sistema, onde todas as decisões são tomadas
- Scheduler: O gerente de recursos, como um diretor de orquestra
- Controller Manager: O executivo, garantindo que tudo esteja no lugar certo
- etcd: O banco de dados, armazenando o estado do cluster
- kubelet: O agente de cada nó, como um agente de segurança
- kube-proxy: O roteador, como um detetive de tráfego

## Docker Swarm: A Colmeia Digital

Enquanto Kubernetes é o leviatã dos mares profundos, Swarm é a colmeia no jardim dos fundos - menor, mais simples, mas não menos fascinante em sua complexidade organizada.





## Conceitos Básicos

- Nodes
- Services
- Tasks
- Stacks

## Comandos Essenciais

```
# Inicializar swarm
docker swarm init

# Criar serviço
docker service create --name web nginx

# Escalar serviço
docker service scale web=3
```

## Comparativo de Orquestradores

### Kubernetes vs Swarm

Aspecto	Kubernetes	Swarm
Complexidade	Alta	Baixa
Escalabilidade	Muito Alta	Moderada
Curva de Aprendizado	Íngreme	Suave
Recursos	Abundantes	Básicos

## Melhores Práticas

### Planejamento

- Arquitetura distribuída
- Estratégia de deployment
- Política de recursos
- Monitoramento

### Segurança

- RBAC
- Network Policies
- Secrets Management
- Container Security

## Troubleshooting

### Problemas Comuns

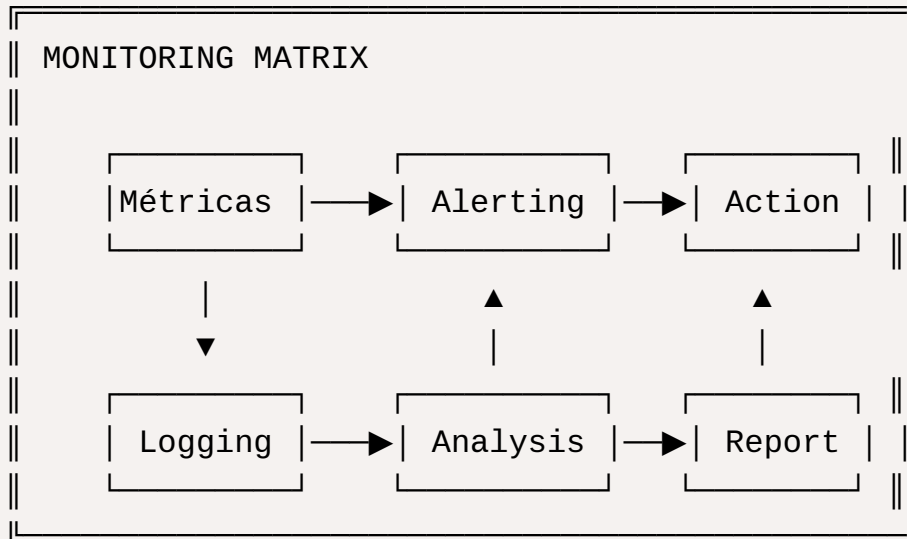
- Node failures

- Network issues
- Resource constraints
- Configuration errors

## **Ferramentas de Debug**

- Logs centralizados
- Métricas
- Tracing
- Health checks

# Container Monitoring: Os Olhos do Sistema



## Métricas Essenciais

### Container Health

```
# CPU Usage
docker stats --format "table {{.Container}}\t{{.CPUPerc}}"

# Memory Consumption
docker stats --format "table {{.Container}}\t{{.MemUsage}}"

# I/O Operations
docker stats --format "table {{.Container}}\t{{.BlockIO}}"

# Detailed Stats
docker stats --no-stream --format \
    "table
```

```
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO}}
```

## Advanced Metrics Collection

### cAdvisor Integration

```
version: '3'
services:
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.47.0
    container_name: cadvisor
    privileged: true
    devices:
      - /dev/kmsg:/dev/kmsg
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
      - /dev/disk/:/dev/disk:ro
    ports:
      - "8080:8080"
```

### Node Exporter Setup

```
version: '3'
services:
  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    command:
      - '--path.procfs=/host/proc'
      - '--path.rootfs=/rootfs'
      - '--path.sysfs=/host/sys'
      - '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($|/)'
```

```
volumes:
  - /proc:/host/proc:ro
  - /sys:/host/sys:ro
  - /:/rootfs:ro
ports:
  - "9100:9100"
```

## Observability Stack

### Prometheus Advanced Configuration

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  - "alert.rules"

scrape_configs:
  - job_name: 'containers'
    static_configs:
      - targets: ['localhost:9090']
    metrics_path: '/metrics'
    relabel_configs:
      - source_labels: [__meta_docker_container_name]
        target_label: container_name
      - source_labels:
          [__meta_docker_container_label_com_docker_compose_service]
        target_label: service_name

  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']

  - job_name: 'node-exporter'
```



```
static_configs:
  - targets: ['node-exporter:9100']
```

## Grafana Dashboards

### Container Overview

```
{
  "dashboard": {
    "panels": [
      {
        "title": "CPU Usage",
        "type": "graph",
        "targets": [
          {
            "expr":
"rate(container_cpu_usage_seconds_total{name=~\"$container\"}
[1m])",
            "legendFormat": "{{name}}"
          }
        ]
      },
      {
        "title": "Memory Usage",
        "type": "graph",
        "targets": [
          {
            "expr":
"container_memory_usage_bytes{name=~\"$container\"}",
            "legendFormat": "{{name}}"
          }
        ]
      }
    ]
  }
}
```

# Advanced Logging Architecture

## Fluentd Configuration

```
<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>

<filter docker.**>
  @type parser
  format json
  key_name log
  reserve_data true
</filter>

<match docker.**>
  @type elasticsearch
  host elasticsearch
  port 9200
  logstash_format true
  logstash_prefix docker
  include_tag_key true
</match>
```

## Vector Pipeline

```
[sources.docker_logs]
type = "docker_logs"
include_containers = ["app*", "web*"]

[transforms.parse_json]
type = "remap"
inputs = ["docker_logs"]
source = '''
. = parse_json!(.message)
```

```
'''
```

```
[sinks.elasticsearch]
type = "elasticsearch"
inputs = ["parse_json"]
endpoint = "http://elasticsearch:9200"
index = "logs-%Y-%m-%d"
```

## Distributed Tracing

### Jaeger Configuration

```
version: '3'
services:
  jaeger:
    image: jaegertracing/all-in-one:latest
    ports:
      - "5775:5775/udp"
      - "6831:6831/udp"
      - "6832:6832/udp"
      - "5778:5778"
      - "16686:16686"
      - "14250:14250"
      - "14268:14268"
      - "14269:14269"
      - "9411:9411"
```

### OpenTelemetry Integration

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
```

```

processors:
  batch:
    timeout: 1s
    send_batch_size: 1024

exporters:
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [jaeger]

```

## Advanced Alerting Strategies

### Alert Manager Configuration

```

global:
  resolve_timeout: 5m
  slack_api_url:
    'https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK'

route:
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  receiver: 'slack-notifications'

receivers:
- name: 'slack-notifications'

```

```
slack_configs:
- channel: '#monitoring'
  title: "{{ range .Alerts }}"{{ .Annotations.summary }}\n{{ end
  }}"
  text: "{{ range .Alerts }}"{{ .Annotations.description }}\n{{
  end }}"
```

## PagerDuty Integration

```
receivers:
- name: 'pagerduty-critical'
  pagerduty_configs:
  - service_key: '<your-pagerduty-key>'
    severity: 'critical'
    description: '{{ template "pagerduty.default.description" .
    }}'
    client: 'monitoring-system'
    client_url: '{{ template "pagerduty.default.clientURL" . }}'
```

## Performance Optimization

### Resource Quotas

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
```

## Monitoring Best Practices

### Retention Policies

```
CREATE RETENTION POLICY "one_month"
ON "docker_metrics"
DURATION 30d
REPLICATION 1
DEFAULT
```

### Downsampling

```
SELECT mean("value")
FROM "cpu_usage"
WHERE time > now() - 7d
GROUP BY time(1h)
```

## Troubleshooting Guide

### Common Scenarios Matrix

Cenário	Sintomas	Métricas Chave	Ação Corretiva
Memory Leak	Crescimento constante de memória	container_memory_usage_bytes	Heap dump e análise
Network Bottleneck	Latência alta	container_network_transmit_bytes_total	Network trace
Disk I/O	Tempo de resposta alto	container_fs_writes_bytes_total	iostat análise
CPU Throttling	Performance degradada	container_cpu_cfs_throttled_seconds_total	Profile CPU


### Debug Toolkit


```
# Container Process Tree
pstree -p $(docker inspect -f '{{.State.Pid}}' container_id)


# Network Connections
nsenter -t $(docker inspect -f '{{.State.Pid}}' container_id) -n
netstat -tupn

# File System Usage
docker run --rm -v /:/host:ro alpine:latest df -h /host

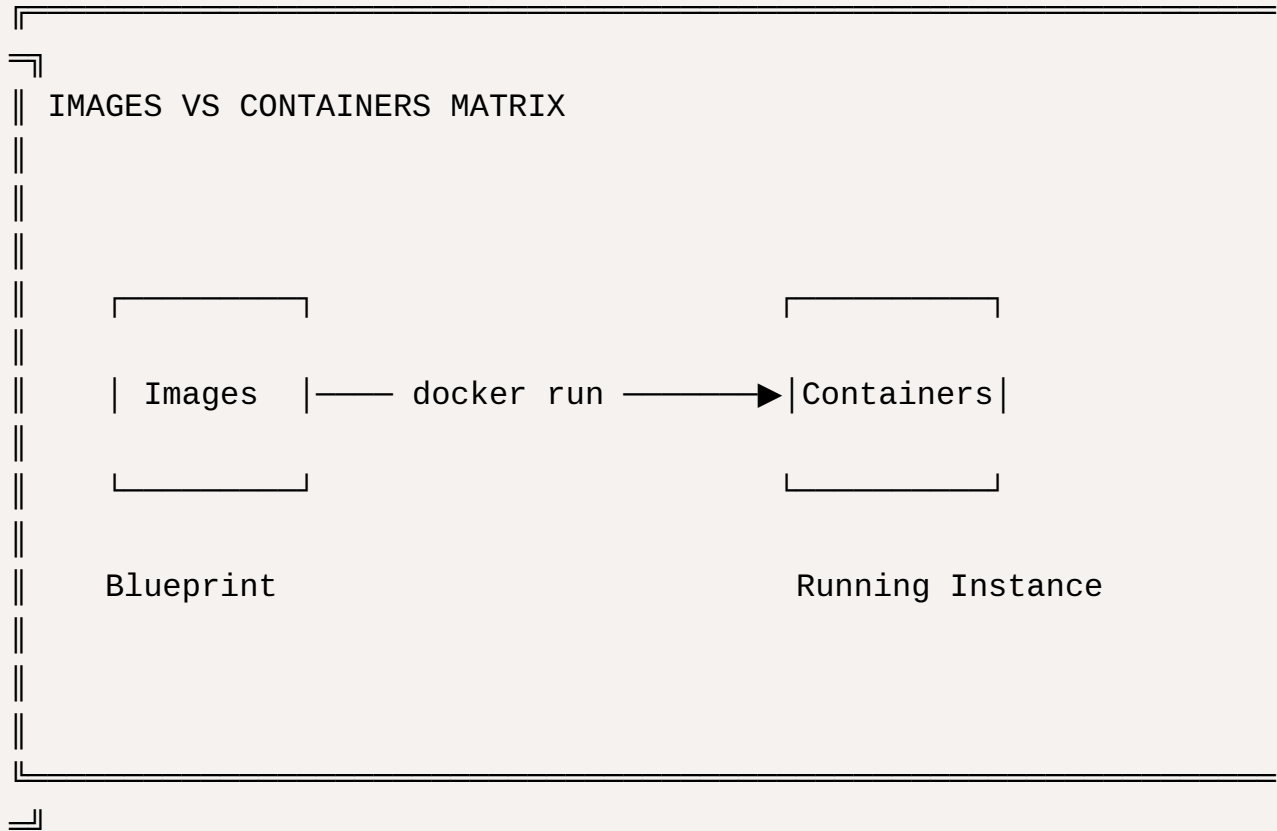
# Stack Trace
docker exec container_id jstack $(pgrep java)
```

 **Monitor-chan diz:** "Métricas são como batimentos cardíacos do sistema - monitore-os ou enfrente o flatline! uwu"

 **Alert-sama avisa:** "Um alerta ignorado hoje é um incidente crítico amanhã! Nya~"

 **Trace-senpai sugere:** "Distributed tracing é como seguir pegadas digitais no cyberspaço! owo"

# Images vs Containers: O Dual Boot do Futuro



## O que você vai aprender

Neste módulo, você mergulhará no universo das imagens e containers Docker, entendendo:

- A diferença fundamental entre imagens e containers
- Como imagens funcionam como blueprints para containers
- O ciclo de vida de containers
- Gerenciamento básico de imagens e containers
- Melhores práticas e padrões comuns



# A Dualidade Fundamental

## Imagens: O Blueprint Imutável

Imagens Docker são como plantas arquitetônicas digitais - templates read-only que contêm:

- Sistema operacional base
- Dependências
- Configurações
- Código da aplicação

## Containers: A Instância Viva

Containers são instâncias em execução de uma imagem, como um prédio construído a partir de uma planta:

- Ambiente isolado e executável
- Estado mutável
- Processos em execução
- Recursos computacionais alocados

## Analogia Matrix

IMAGEM	CONTAINER
==	==
- Blueprint	- Instância em execução
- Imutável	- Mutável
- Template	- Processo vivo
- Compartilhável	- Isolado

|| - Armazenada em disco - Executa em memória ||

## Roadmap de Aprendizado

### 1. Fundamentos de Imagens ([Fundamentos de Imagens Docker](#) )

- Conceitos básicos
- Componentes principais
- Características essenciais

### 2. Anatomia de uma Imagem ([Anatomia de uma Imagem Docker](#) )

- Sistema de layers
- Manifests e configurações
- Formato e especificações

### 3. Gerenciamento de Imagens ([Gerenciamento de Imagens Docker](#) )

- Comandos básicos
- Tags e versionamento
- Registries e distribuição

### 4. Operações com Containers ([Operações com Containers: O Manual do Operador](#) )

- Ciclo de vida
- Comandos essenciais
- Estados e transições

### 5. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) )

- Guidelines de segurança
- Otimizações

- Padrões recomendados

## 6. Laboratório Prático ([Laboratório Prático: Mão na Massa!](#) )

- Exercícios guiados
- Casos de uso reais
- Desafios práticos

## 7. Guia de Troubleshooting ([Guia de Troubleshooting: Debugando na Matrix](#) )

- Problemas comuns
- Soluções
- Debugging


## Quick Start



```
# Baixar uma imagem
docker pull nginx:latest

# Criar e executar um container
docker run -d -p 80:80 nginx:latest

# Listar containers em execução
docker ps

# Parar o container
docker stop <container-id>
```

 **Image-chan diz:** "Pense em mim como seu template perfeito - imutável e confiável! uwu"

 **Container-kun avisa:** "E eu sou a manifestação viva do seu código - ativo e dinâmico! 

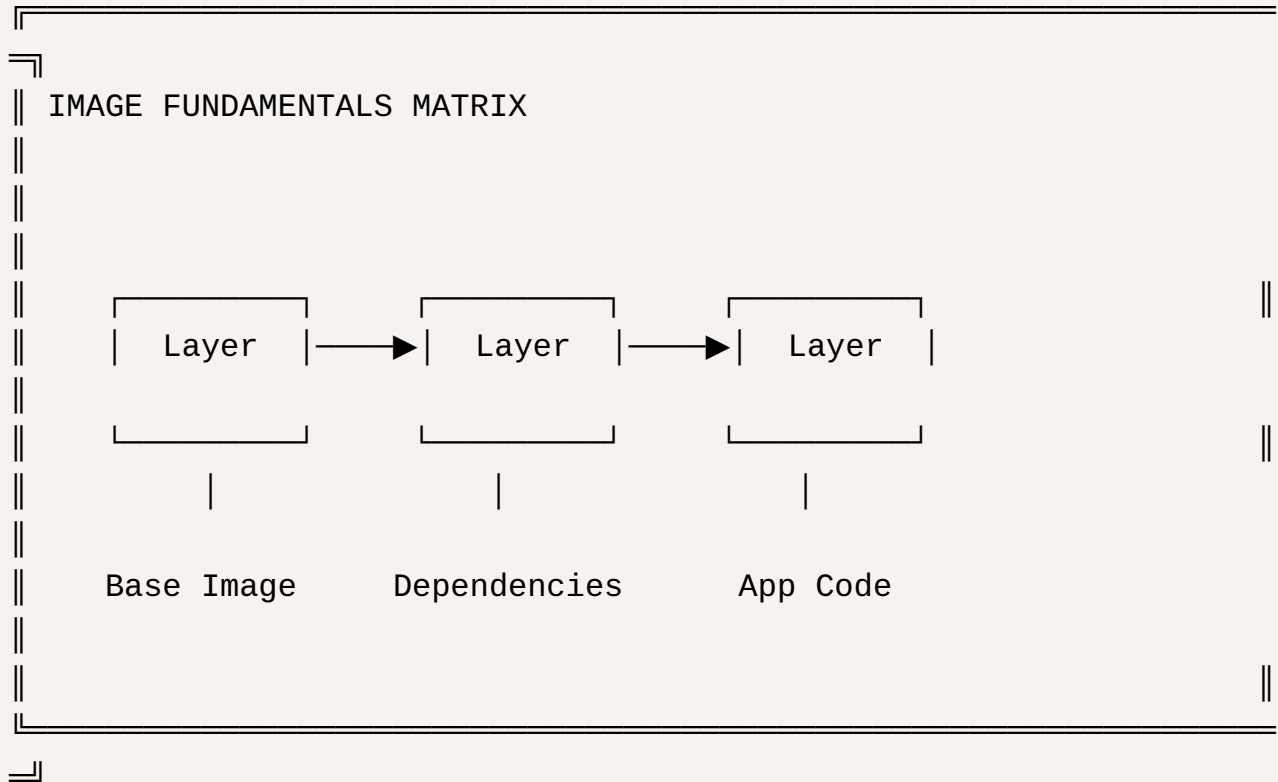
## Próximos Passos 🎮

Explore cada subtópico na ordem sugerida para uma jornada de aprendizado estruturada. Comece com os Fundamentos de Imagens ([Fundamentos de Imagens Docker](#) 📦) para construir uma base sólida.



**Sensei's Corner:** "A jornada de mil containers começa com uma única imagem!"

# Fundamentos de Imagens Docker



## Conceitos Fundamentais 🧠

### O que é uma Imagem Docker?

Uma imagem Docker é um pacote leve, autônomo e executável que inclui tudo necessário para rodar uma aplicação:

- Código
- Runtime
- Bibliotecas
- Variáveis de ambiente

- Arquivos de configuração

## Características Principais

### CARACTERÍSTICAS ESSENCIAIS

- ♦ Imutabilidade
- ♦ Sistema de Camadas
- ♦ Compartilhamento de Recursos
- ♦ Portabilidade
- ♦ Versionamento

## Anatomia de uma Imagem

### Sistema de Camadas

#### 1. Base Layer

- Sistema operacional mínimo
- Bibliotecas essenciais

#### 2. Camadas Intermediárias

- Dependências
- Ferramentas
- Configurações

#### 3. Camada de Aplicação

- Código fonte
- Assets
- Configurações específicas

# Tipos de Imagens

## 1. Imagens Base

- Alpine
- Ubuntu
- Debian
- Scratch

## 2. Imagens de Runtime

- Node.js
- Python
- Java
- Go

## 3. Imagens de Aplicação

- Nginx
- MongoDB
- Redis
- PostgreSQL

# Nomenclatura e Tags

```
|| ANATOMIA DE UM NOME DE IMAGEM ||
||
|| registry.example.com/organization/image-name:tag ||
||   └─ Registry ─┘   └─ Owner ─┘   └─ Name ─┘   └─ Tag ─┘   ||
```

## Convenções de Tags

- `latest`: versão mais recente
- `1.0.0`: versão específica
- `stable`: versão estável
- `alpine`: baseada em Alpine Linux

## Comandos Essenciais

```
# Listar imagens locais
docker images

# Baixar uma imagem
docker pull nginx:latest

# Inspecionar uma imagem
docker inspect nginx:latest

# Remover uma imagem
docker rmi nginx:latest

# Mostrar histórico de camadas
docker history nginx:latest
```

## Boas Práticas

### DO's

- Use tags específicas
- Minimize o tamanho das imagens



- Documente dependências
- Utilize multi-stage builds
- Mantenha imagens atualizadas

## DON'Ts ❌

- Evite `latest` em produção
- Não armazene secrets
- Não instale ferramentas desnecessárias
- Não use imagens não oficiais sem verificação
- Não ignore vulnerabilidades

## Waifu Tips 💡

⚠️ **Image-chan diz:** "Mantenha suas camadas organizadas como um bento box perfeito! 🍱"

⚠️ **Security-chan avisa:** "Sempre verifique a origem das suas imagens base! 🔍"

## Próximos Passos 🎯

1. Anatomia de uma Imagem ([Anatomia de uma Imagem Docker](#) 🧐)
2. Gerenciamento de Imagens ([Gerenciamento de Imagens Docker](#) 🎮)
3. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) 🎯)

## Checkpoint ✅

Você agora entende:

- [x] O que são imagens Docker
- [x] Sistema de camadas
- [x] Tipos de imagens
- [x] Nomenclatura e tags
- [x] Comandos básicos
- [x] Boas práticas



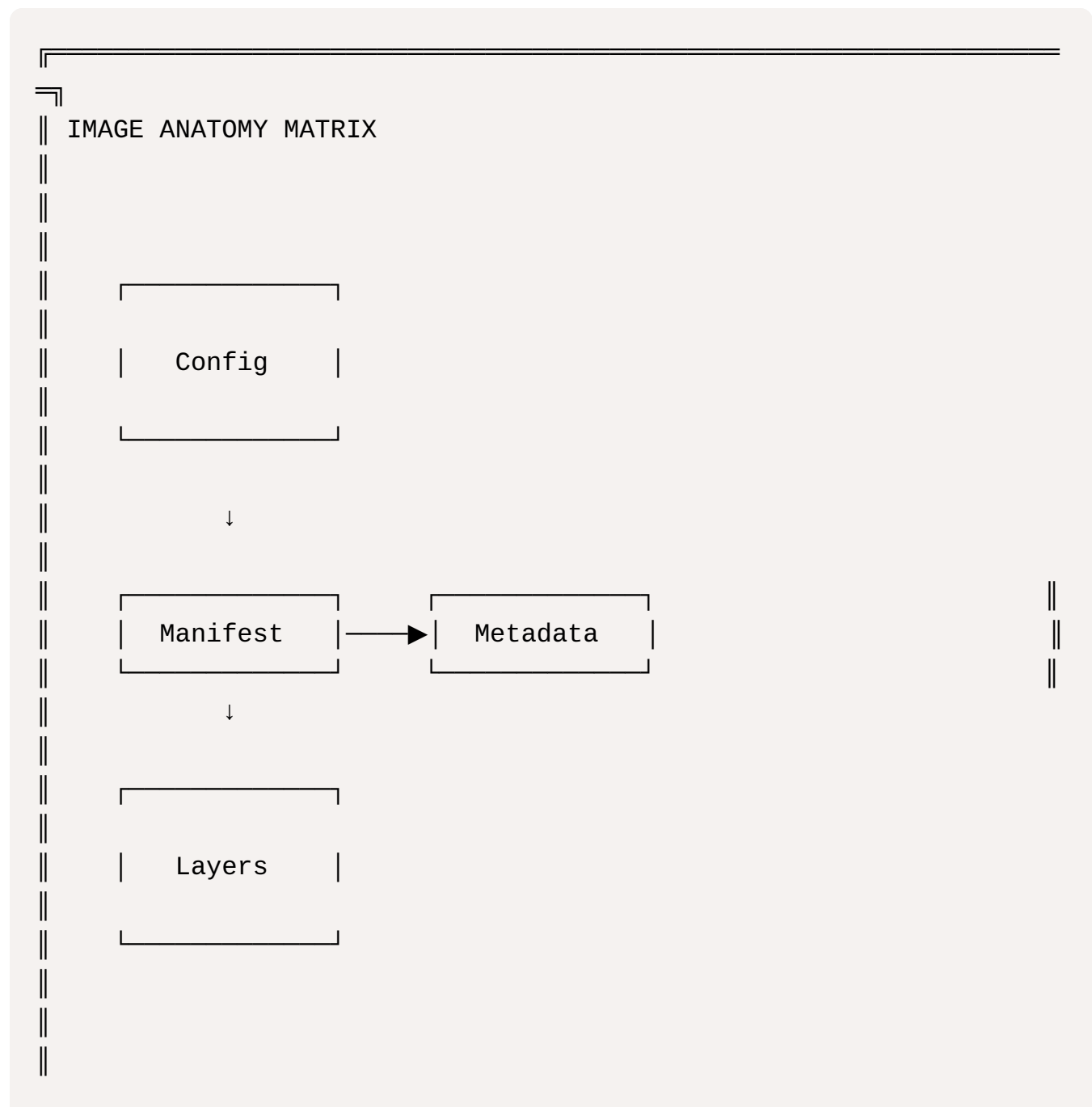
**Sensei's Corner:** "Uma imagem bem construída é como um kata bem executado - precisa, eficiente e elegante."

# Anatomia de uma Imagem Docker



Uma imagem Docker é composta por várias camadas sobrepostas que formam um sistema de arquivos unificado. Vamos dissecar cada componente.

## Visão Geral da Estrutura





## 1. Arquivo de Configuração

O arquivo de configuração é um documento JSON que define o comportamento da imagem.

### Componentes Principais

```
{
  "config": {
    "Hostname": "",
    "Domainname": "",
    "User": "",
    "ExposedPorts": {
      "80/tcp": {}
    },
    "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": [
      "nginx",
      "-g",
      "daemon off;"
    ],
    "WorkingDir": "/",
    "Volumes": null
  }
}
```

### Elementos Essenciais

- **ExposedPorts**: Portas que o container expõe

- **Env**: Variáveis de ambiente padrão
- **Cmd**: Comando padrão executado no container
- **WorkingDir**: Diretório de trabalho inicial
- **User**: Usuário padrão do container

## 2. Manifesto da Imagem

O manifesto é um documento JSON que descreve a composição da imagem.

### Estrutura do Manifesto

```
{
  "schemaVersion": 2,
  "mediaType":
"application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 1234,
    "digest": "sha256:a1b2c3..."
  },
  "layers": [
    {
      "mediaType":
"application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 5678,
      "digest": "sha256:d4e5f6..."
    }
  ]
}
```

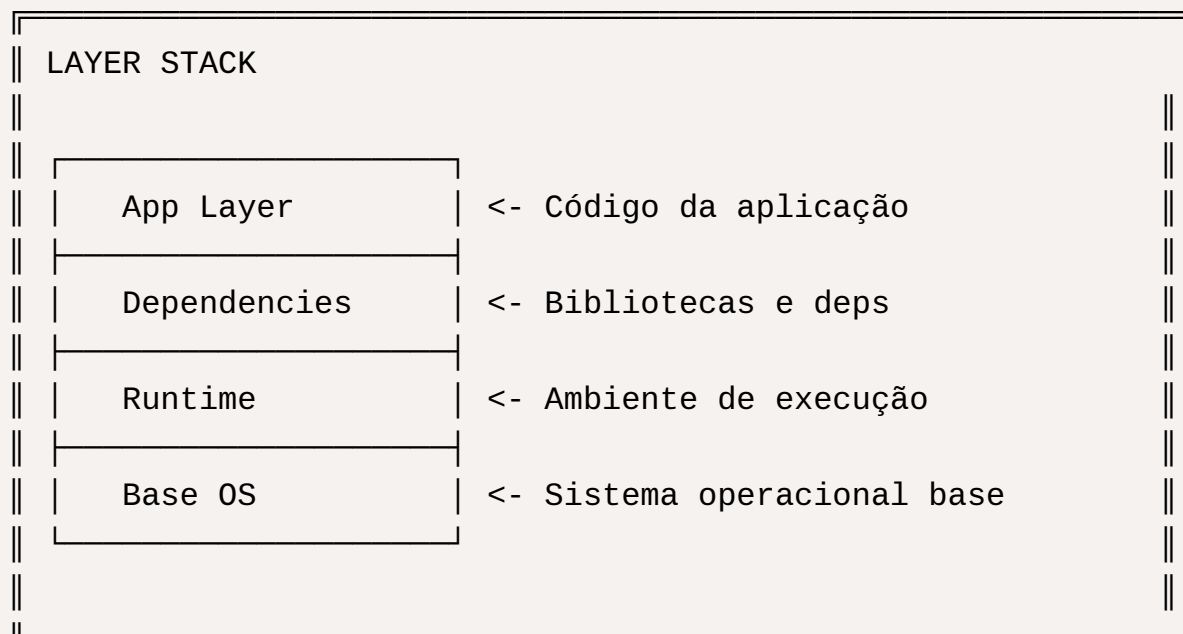
### Elementos do Manifesto

- **schemaVersion**: Versão do formato do manifesto
- **config**: Referência ao arquivo de configuração

- **layers**: Lista de camadas que compõem a imagem
- **digest**: Hash SHA256 único da camada

### 3. Sistema de Camadas

As camadas são organizadas em uma estrutura de pilha, onde cada camada representa uma modificação no sistema de arquivos.



#### Características das Camadas

##### Camadas Read-Only

- Imutáveis após criação
- Compartilhadas entre containers
- Armazenadas em cache localmente

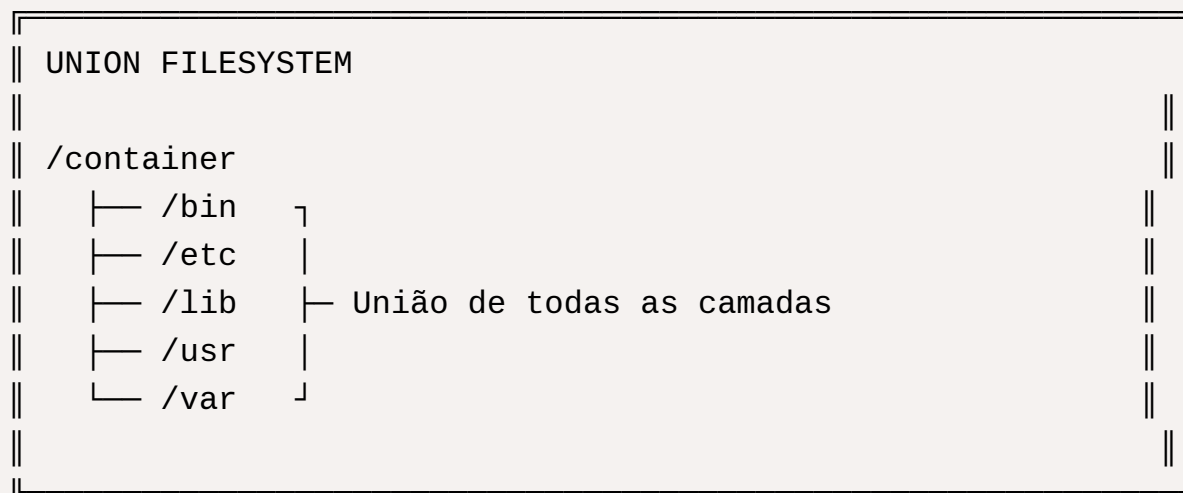
##### Camada Read-Write

- Criada ao iniciar um container
- Específica para cada container

- Contém todas as alterações em runtime

## 4. Sistema de Arquivos Union

O Docker utiliza um sistema de arquivos em camadas que combina todas as camadas em uma única visão.



### Funcionamento

1. Cada camada representa um conjunto de diferenças
2. As camadas são montadas sobrepostas
3. Arquivos mais recentes têm precedência
4. Deleções são marcadas com whiteouts

## 5. Comandos de Inspeção

### Análise de Imagens

```
# Metadata da imagem
docker inspect nginx:latest

# Histórico de camadas
docker history nginx:latest
```

```
# Digest da imagem
docker images --digests nginx

# Exportar imagem
docker save nginx:latest > nginx.tar
```

## Waifu Tips 💡

⚠️ **Layer-chan diz:** "Cada camada é como um capítulo da história do seu container. Mantenha-a interessante, mas concisa! 📖"

⚠️ **Storage-chan avisa:** "Cuidado com o copy-on-write! Muitas escritas podem impactar a performance! 🚀"

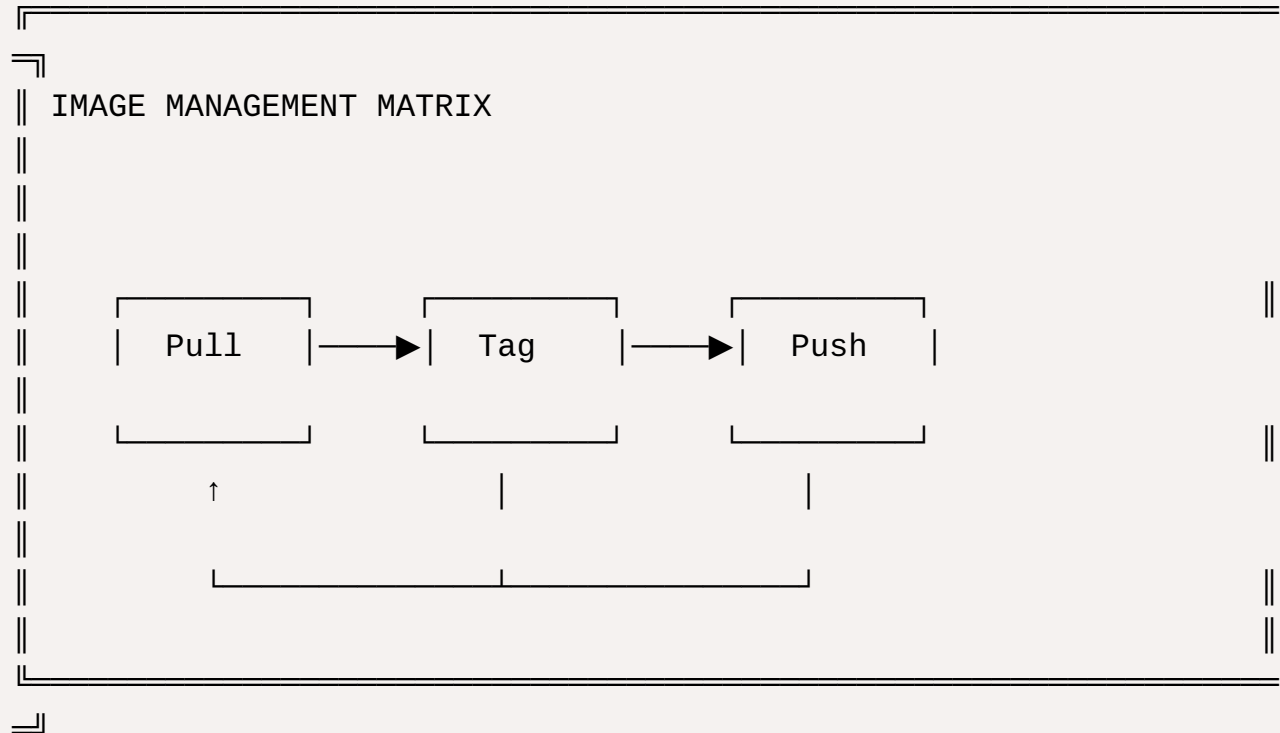
## Próximos Passos

1. Gerenciamento de Imagens ([Gerenciamento de Imagens Docker](#) 🎮)
2. Operações com Containers ([Operações com Containers: O Manual do Operador](#) 🎮)
3. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) 🎯)



# Gerenciamento de Imagens

## Docker



## Comandos Essenciais

### Gerenciamento Básico

```
# Listar imagens
docker images

# Baixar imagem
docker pull nginx:latest

# Remover imagem
docker rmi nginx:latest
```

```
# Remover todas imagens não utilizadas
docker image prune
```

## Tagging e Versionamento

```
# Criar tag
docker tag nginx:latest meu-registry.com/nginx:v1.0

# Remover tag
docker rmi meu-registry.com/nginx:v1.0

# Listar tags de uma imagem
docker image inspect nginx --format='{{.RepoTags}}'
```

## Sistema de Tags

The diagram illustrates the structure of a Docker image tag. It features a large rectangular box with a double border. Inside the box, the text "TAG CONVENTIONS" is at the top. Below it, the example tag "registry.example.com/org/app:1.0.0-alpine" is shown. Underneath the tag, four brackets with labels identify its parts: "Registry" for "registry.example.com", "Name" for "org/app", "Tag" for "1.0.0", and "Variant" for "alpine".

```
graph TD; subgraph Box [ ]; TAG[registry.example.com/org/app:1.0.0-alpine]; TAG --- REG[Registry]; TAG --- NAME[Name]; TAG --- TAG_PART[Tag]; TAG --- VARIANT[Variant]; end
```

## Convenções Comuns

- latest: versão mais recente
- 1.0.0: versão semântica
- stable: versão estável
- dev: versão desenvolvimento
- alpine: variante específica

## Registries e Distribuição

### Docker Hub

```
# Login no Docker Hub
docker login

# Push para Docker Hub
docker push username/app:tag

# Pull do Docker Hub
docker pull username/app:tag
```

### Registry Privado

```
# Login em registry privado
docker login meu-registry.com

# Push para registry privado
docker push meu-registry.com/app:tag

# Pull de registry privado
docker pull meu-registry.com/app:tag
```

## Otimização e Limpeza

### Limpeza de Sistema

```
# Remover imagens não utilizadas
docker image prune -a

# Remover imagens dangling
docker image prune
```

```
# Limpar todo o sistema  
docker system prune
```

## Otimização de Espaço

```
# Verificar uso de espaço  
docker system df  
  
# Análise detalhada  
docker system df -v
```

## Boas Práticas ✨

### DO's ✓

- Use tags específicas em produção
- Mantenha um registro de versões
- Documente variantes de imagens
- Implemente política de retenção
- Automatize o processo de build/push

### DON'Ts ✗

- Evite usar `latest` em produção
- Não acumule imagens antigas
- Não armazene dados sensíveis
- Não ignore vulnerabilidades
- Não deixe imagens sem tag

## Automação e CI/CD 🤖

## GitHub Actions Example

```
steps:
  - uses: actions/checkout@v2

  - name: Login to Registry
    run: docker login -u ${ secrets.DOCKER_USER }} -p ${ secrets.DOCKER_TOKEN }}

  - name: Build and Push
    run: |
      docker build -t app:${ github.sha }} .
      docker push app:${ github.sha }}
```

## Waifu Tips 💡

⚠️ **Registry-chan diz:** "Mantenha suas tags organizadas como uma coleção de mangás! 📚"

⚠️ **Space-chan avisa:** "Não deixe imagens antigas ocuparem espaço! Faça limpeza regularmente! 🧹"

## Troubleshooting 🔧

### Problemas Comuns

#### 1. Pull Fails

```
# Verificar conectividade
docker info




# Limpar cache DNS
docker system prune --volumes
```


## 2. Push Errors

```
# Verificar autenticação
docker login

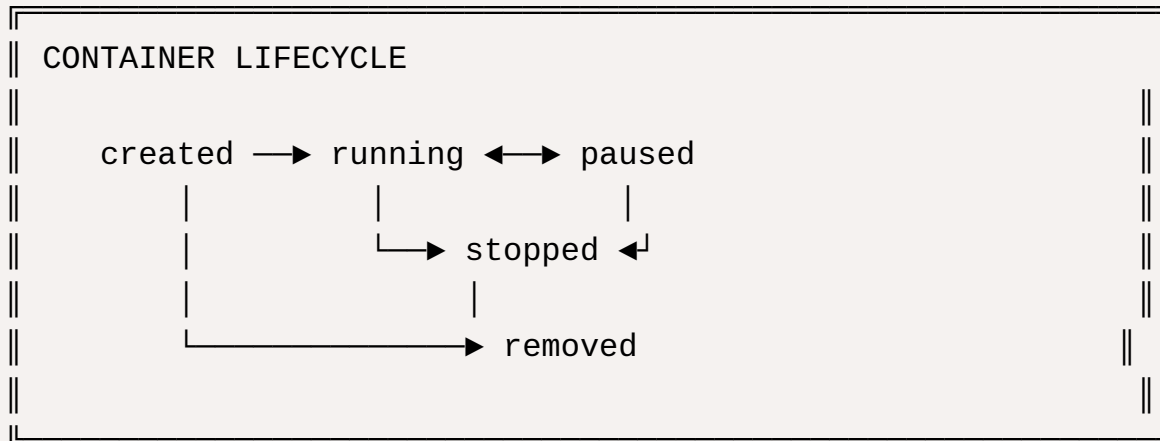
# Verificar permissões
docker push app:tag
```

## Próximos Passos

1. Operações com Containers ([Operações com Containers: O Manual do Operador](#) )
2. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) )
3. Laboratório Prático ([Laboratório Prático: Mão na Massa!](#) )

 **Sensei's Note:** "Um bom gerenciamento de imagens é como manter um dojo organizado - cada coisa em seu lugar."

# Operações com Containers: O Manual do Operador 🎮



## Operações Básicas 🛠️

### Ciclo de Vida

```
# Criar container
docker create --name meu-app nginx

# Iniciar container
docker start meu-app

# Parar container
docker stop meu-app

# Remover container
docker rm meu-app
```

### Execução e Acesso

```
# Executar novo container
docker run -d --name web nginx
```

```
# Executar comando em container
docker exec -it web bash

# Visualizar logs
docker logs web

# Copiar arquivos
docker cp arquivo.txt web:/app/
```

## Gerenciamento de Estado

### Status e Inspeção

```
# Listar containers
docker ps -a

# Inspecionar container
docker inspect web

# Estatísticas em tempo real
docker stats web

# Processos em execução
docker top web
```

### Controle de Recursos

```
# Limitar CPU e memória
docker run -d \
  --cpus=".5" \
  --memory="512m" \
  --name app \
  nginx
```



```
# Atualizar limites
docker update --cpus=".8" app
```

## Network Operations

### Conectividade

```
# Criar rede
docker network create minha-rede

# Conectar container
docker network connect minha-rede web

# Desconectar container
docker network disconnect minha-rede web

# Inspecionar rede
docker network inspect minha-rede
```

## Volume Operations

### Persistência de Dados

```
# Criar volume
docker volume create meus-dados

# Montar volume
docker run -d \
  -v meus-dados:/data \
  --name db \
  postgres

# Listar volumes
docker volume ls
```

```
# Remover volume
docker volume rm meus-dados
```

## Troubleshooting Matrix 🔍

Problema	Comando	Solução
Container não inicia	<code>docker logs</code>	Verificar logs de erro
Container travado	<code>docker restart</code>	Reiniciar container
Memória alta	<code>docker stats</code>	Ajustar limites
Rede instável	<code>docker network inspect</code>	Verificar conectividade

## Batch Operations 🚀

### Operações em Massa

```
# Parar todos containers
docker stop $(docker ps -q)

# Remover containers parados
docker container prune

# Remover tudo não usado
docker system prune -a

# Backup de volumes
docker run --rm \
  -v volume_name:/source \
  -v $(pwd):/backup \
  alpine tar czf /backup/volume_backup.tar.gz /source
```



## Health Checks



### Monitoramento de Saúde

```
# Container com health check
docker run -d \
  --health-cmd="curl -f http://localhost/ || exit 1" \
  --health-interval=5s \
  --name web \
  nginx

# Verificar status
docker inspect --format='{{.State.Health.Status}}' web
```

## Waifu Operation Tips

 **Container-chan diz:** "Mantenha seus containers leves e eficientes! 

 **Resource-sama avisa:** "Sempre defina limites de recursos para evitar surpresas! 

## Debug Commands

### Ferramentas de Debug

```
# Logs em tempo real
docker logs -f web

# Histórico de eventos
docker events --filter container=web

# Dump de estado
docker inspect web > debug.json
```

```
# Trace de rede
docker run --net container:web nicolaka/netshoot
```

## Automation Scripts

### Exemplos de Automação




```
#!/bin/bash
# Restart containers mais velhos que 7 dias
docker ps -q | xargs docker inspect \
  --format='{{.Id}} {{.State.StartedAt}}' | \
  awk '{if ($2 <= "$(date -d'7 days ago' -Ins --utc)') print $1}' | \
  xargs docker restart
```

## Checkpoint

Você agora sabe:

- [x] Gerenciar ciclo de vida
- [x] Controlar recursos
- [x] Operar redes
- [x] Gerenciar volumes
- [x] Debugar problemas
- [x] Automatizar operações

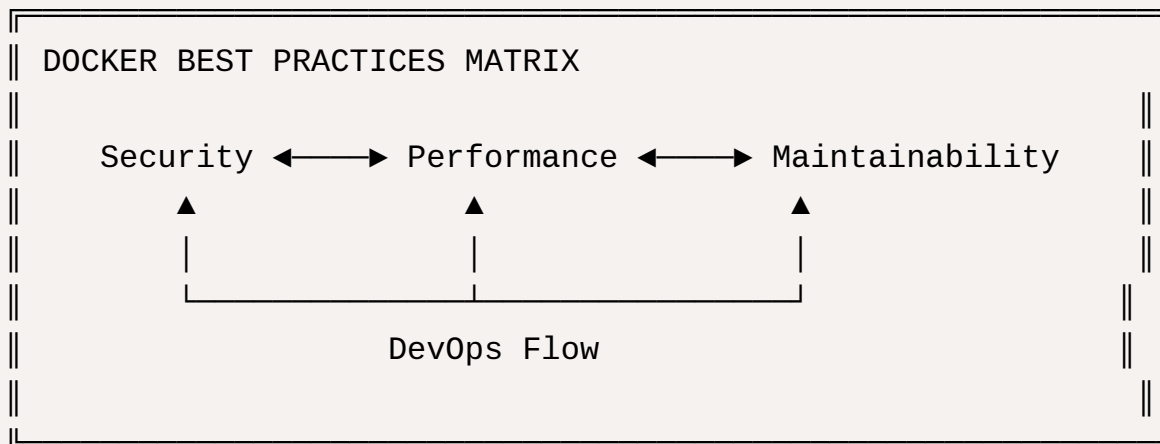
## Próximos Passos

1. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) )
2. Laboratório Prático ([Laboratório Prático: Mão na Massa!](#) )
3. Guia de Troubleshooting ([Guia de Troubleshooting: Debugando na Matrix](#) )




**Sensei's Note:** "A maestria em operações de containers vem da prática constante e da atenção aos detalhes."

# Melhores Práticas Docker: O Caminho do Mestre




## Dockerfile Best Practices

### Otimização de Imagens

```
#  Multi-stage build
FROM node:alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

### Layer Optimization

```
#  Combine RUN commands
RUN apt-get update && \
    apt-get install -y \
```

```
package1 \  
package2 && \  
rm -rf /var/lib/apt/lists/*
```

```
# ❌ Avoid multiple RUN commands  
RUN apt-get update  
RUN apt-get install package1  
RUN apt-get install package2
```

## Security Guidelines

### Container Hardening

```
# ✅ Use non-root user  
FROM alpine  
RUN adduser -D appuser  
USER appuser  
  
# ✅ Read-only root filesystem  
docker run --read-only nginx
```

### Secrets Management

```
# ✅ Use Docker secrets  
docker secret create app_secret secret.txt  
docker service create \  
  --secret app_secret \  
  myapp  
  
# ❌ Avoid environment variables for secrets  
docker run -e "API_KEY=secret" myapp
```

## Performance Optimization

### Resource Management

```
# ✓ Set resource limits
docker run \
  --cpus=".5" \
  --memory="512m" \
  --memory-swap="1g" \
  nginx
```

## Networking

```
# ✓ Use user-defined networks
docker network create --driver overlay mynet

# ✓ Enable DNS caching
docker run --dns-opt="ndots:1" nginx
```

## Development Workflow

### Docker Compose

```
# ✓ docker-compose.yml
version: '3.8'
services:
  app:
    build:
      context: .
      target: development
    volumes:
      - ./app
      - /app/node_modules
    environment:
      - NODE_ENV=development
```

## Testing

```
# ✓ Dedicated test container
```



```
docker-compose -f docker-compose.test.yml up
```

## Production Deployment 🌟

### Health Checks

```
# ✅ Add HEALTHCHECK
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

### Logging

```
# ✅ Configure logging
docker run \
  --log-driver json-file \
  --log-opt max-size=10m \
  --log-opt max-file=3 \
  nginx
```

## CI/CD Integration 🔄

### Build Pipeline

```
# ✅ GitHub Actions example
name: Docker Build



on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Build and test
        run: |
```

```
docker build -t myapp:test .  
docker run myapp:test npm test
```

## Monitoring & Maintenance

### Container Health

```
#  Regular health checks  
docker inspect --format='{{.State.Health.Status}}' container_name  
  
#  Resource monitoring  
docker stats --format "table  
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

## DO's and DON'Ts Matrix ✨

### DO's

- |                            |
|----------------------------|
| 1. Use multi-stage builds  |
| 2. Set resource limits     |
| 3. Implement health checks |
| 4. Use .dockerignore       |
| 5. Version control images  |

### DON'Ts

- |                            |
|----------------------------|
| 1. Run as root             |
| 2. Store secrets in images |
| 3. Use latest tag          |
| 4. Ignore security scans   |
| 5. Skip health checks      |

## Waifu Best Practice Tips 💡

⚠️ Security-chan diz: "Nunca confie em imagens não verificadas! 🔍"

⚠️ Performance-sama avisa: "Containers leves são containers felizes! 🎈"

## Checklist de Qualidade ✅

### Antes do Deploy


1. ☐ Imagem otimizada
2. ☐ Security scan realizado
3. ☐ Resources limits definidos
4. ☐ Health checks implementados
5. ☐ Logs configurados
6. ☐ Backups planejados
7. ☐ Monitoring setup

## Troubleshooting Guide 🛠️

### Common Issues


```
# ✅ Debug container
docker logs container_name

# ✅ Interactive debug
docker exec -it container_name sh
```

```
#  Network debug  
docker network inspect bridge
```

## Próximos Passos

1. Laboratório Prático ([Laboratório Prático: Mão na Massa!](#) )
2. Guia de Troubleshooting ([Guia de Troubleshooting: Debugando na Matrix](#) )
- 3.

 **Sensei's Note:** "As melhores práticas são como um kata - devem ser praticadas até se tornarem naturais."

# Laboratório Prático: Mão na Massa!

```
|| DOCKER LAB SIMULATOR v1.0 ||  
||  
|| STATUS: INICIANDO SIMULAÇÃO... ||  
|| DIFICULDADE: PROGRESSIVA ||  
|| OBJETIVO: DOMINAR A MATRIX DOS CONTAINERS ||  
||
```

## Lab 1: Primeira Aplicação

### Objetivo

Criar uma aplicação web simples usando Docker

### Passos

#### 1. Criar Dockerfile

```
FROM python:3.9-slim  
WORKDIR /app  
COPY app.py requirements.txt ./  
RUN pip install -r requirements.txt  
CMD ["python", "app.py"]
```

#### 2. Criar Aplicação

```
from flask import Flask  
app = Flask(__name__)  
  
@app.route('/')
```

```
def hello():  
    return "Olá, Docker!"  
  
if __name__ == '__main__':  
    app.run(host='0.0.0.0')
```

### 3. Build & Run

```
# Build da imagem  
docker build -t minha-app:v1 .  
  
# Executar container  
docker run -p 5000:5000 minha-app:v1
```

## Lab 2: Multi-Container Setup 🧑🏻💻

### Objetivo

Criar aplicação com frontend, backend e banco de dados

### Docker Compose

```
version: '3.8'  
services:  
  frontend:  
    build: ./frontend  
    ports:  
      - "80:80"  
    depends_on:  
      - backend  
  
  backend:  
    build: ./backend  
    environment:  
      - DB_HOST=db  
    depends_on:
```

```
- db

db:
  image: postgres:13
  volumes:
    - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

## Teste de Integração

```
# Subir ambiente
docker-compose up -d

# Verificar logs
docker-compose logs -f

# Testar conexões
curl localhost/api/health
```

## Lab 3: Debugging & Troubleshooting

### Cenário 1: Container Crashando

```
# Investigar logs
docker logs container_id

# Acessar container
docker exec -it container_id sh

# Verificar recursos
docker stats container_id
```

### Cenário 2: Problemas de Rede

```
# Criar rede
docker network create mynet

# Inspecionar rede
docker network inspect mynet

# Testar conectividade
docker run --network mynet busybox ping service_name
```

## Lab 4: Otimização de Imagem

### Antes

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
RUN apt-get install -y python3-pip
COPY . /app
RUN pip install -r requirements.txt
CMD ["python3", "app.py"]
```

### Depois

```
FROM python:3.9-alpine
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python3", "app.py"]
```

## Lab 5: Segurança & Monitoramento

### Security Scan

```
# Scan de vulnerabilidades
docker scan minha-app:v1
```



```
# Análise de configuração
docker bench security
```

## Monitoring Setup

```
version: '3.8'
services:
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
```

## Desafios Extras 🏆

### Challenge 1: High Availability

- Implementar load balancing
- Configurar health checks
- Simular failover

### Challenge 2: CI/CD Pipeline

```
# .github/workflows/docker.yml
name: Docker CI

on: [push]

jobs:
  build:
```

```
runs-on: ubuntu-latest
steps:
  - uses: actions/checkout@v2
  - name: Build
    run: docker build -t app:test .
  - name: Test
    run: docker run app:test pytest
```

## Waifu Lab Assistant Tips 💡

⚠️ Debug-chan diz: "Quando tudo der errado, `docker logs` é seu melhor amigo!" 🔍

⚠️ Security-sama alerta: "Nunca se esqueça de remover as credenciais dos logs!" 🔒

## Checklist de Conclusão ✅

### Lab 1

- ☐ Dockerfile criado
- ☐ Aplicação rodando
- ☐ Portas mapeadas

### Lab 2

- ☐ Compose configurado
- ☐ Serviços comunicando
- ☐ Volumes persistentes

### Lab 3

- ☐ Debug realizado
- ☐ Logs analisados
- ☐ Rede configurada

## Lab 4

- ☐ Imagem otimizada
- ☐ Tamanho reduzido
- ☐ Build mais rápido

## Lab 5

- ☐ Security scan
- ☐ Monitoring ativo
- ☐ Alertas configurados

## Próximos Passos

1. Guia de Troubleshooting ([Guia de Troubleshooting: Debugando na Matrix](#) )
2. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) )
- 3.



**Sensei's Note:** "A prática leva à perfeição, mas backups levam à paz de espírito."

# Guia de Troubleshooting: Debugando na Matrix

```
|| DOCKER DEBUG MATRIX v1.0 ||  
||  
|| STATUS: SCANNING FOR ISSUES... ||  
|| LEVEL: FROM BASIC TO ADVANCED ||  
|| MISSION: RESOLVER PROBLEMAS COMO UM PRO ||  
||
```

## Problemas Comuns e Soluções

### 1. Container não Inicia

```
# Verificar logs  
docker logs container_id  
  
# Verificar status  
docker ps -a  
  
# Inspecionar configuração  
docker inspect container_id
```

### 2. Problemas de Rede

```
# Verificar redes  
docker network ls  
  
# Testar conectividade  
docker run --rm busybox ping container_name
```

```
# Inspecionar rede
docker network inspect bridge
```

### 3. Problemas de Volume

```
# Listar volumes
docker volume ls

# Inspecionar volume
docker volume inspect volume_name

# Verificar permissões
ls -la /var/lib/docker/volumes/
```

## Ferramentas de Debug Avançadas

### Docker Stats

```
# Monitorar recursos
docker stats

# Limitar recursos
docker run --memory=512m --cpu-shares=512 image_name
```

### Docker Events

```
# Monitorar eventos em tempo real
docker events

# Filtrar eventos
docker events --filter 'type=container'
```

### Logging Avançado

```
# Logs com timestamp
docker logs --timestamps container_id
```

```
# Logs em tempo real
docker logs --follow container_id

# Últimas N linhas
docker logs --tail 100 container_id
```

## Técnicas de Investigação

### 1. Container Deep Dive

```
# Acessar container
docker exec -it container_id sh

# Processos rodando
docker top container_id

# Histórico de comandos
docker history image_name
```

### 2. Network Debugging

```
# DNS lookup
docker run --rm busybox nslookup container_name

# Trace route
docker run --rm busybox traceroute host.docker.internal

# Port mapping
docker port container_id
```

### 3. Storage Analysis

```
# Disk usage
docker system df
```

```
# Clean up
docker system prune -a

# Volume cleanup
docker volume prune
```

## Waifu Debug Tips 💡

⚠️ **Debug-chan diz:** "Logs são como pegadas digitais - siga-as e encontrará a verdade! 🔍"

⚠️ **Network-sama alerta:** "Sempre verifique suas configurações de rede duas vezes! 🌐"

## Checklist de Debug ✅

### Básico

- ☐ Verificar logs do container
- ☐ Confirmar status do container
- ☐ Checar configurações de rede
- ☐ Validar volumes montados

### Avançado

- ☐ Analisar métricas de performance
- ☐ Investigar eventos do sistema
- ☐ Verificar segurança
- ☐ Validar recursos disponíveis

## Matriz de Decisão

Sintoma	Primeira Ação	Próxima Ação	Solução Comum
Container Crashando	<code>docker logs</code>	<code>docker inspect</code>	Verificar recursos
Rede Lenta	<code>docker network inspect</code>	<code>ping test</code>	Ajustar DNS
Disco Cheio	<code>docker system df</code>	<code>docker system prune</code>	Limpar volumes
CPU Alta	<code>docker stats</code>	<code>top</code> no container	Limitar recursos

## Comandos de Emergência

### Reset Rápido

```
# Parar todos containers
docker stop $(docker ps -q)

# Remover containers parados
docker container prune

# Limpar sistema
docker system prune -af
```




### Backup de Emergência


```
# Backup de volume
docker run --rm -v volume_name:/source -v $(pwd):/backup alpine
tar -czf /backup/volume_backup.tar.gz -C /source .
```



```
# Backup de container  
docker commit container_id backup_image
```

## Próximos Passos

1. Laboratório Prático ([Laboratório Prático: Mão na Massa!](#) )
2. Melhores Práticas ([Melhores Práticas Docker: O Caminho do Mestre](#) )
3. Monitoramento Avançado ([Container Monitoring: Os Olhos do Sistema](#) )

 **Sensei's Note:** "O melhor debug é aquele que você não precisa fazer - monitore proativamente!"

# Arquitetura Docker: O Blueprint da Matrix

DOCKER ARCHITECTURE BLUEPRINT

Client → Docker Host → Container Runtime → Registry

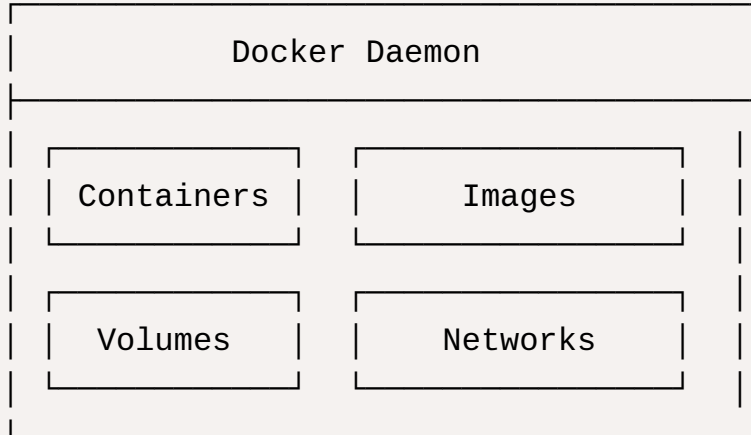
"A arquitetura que sustenta o multiverso dos containers"

## Componentes Principais

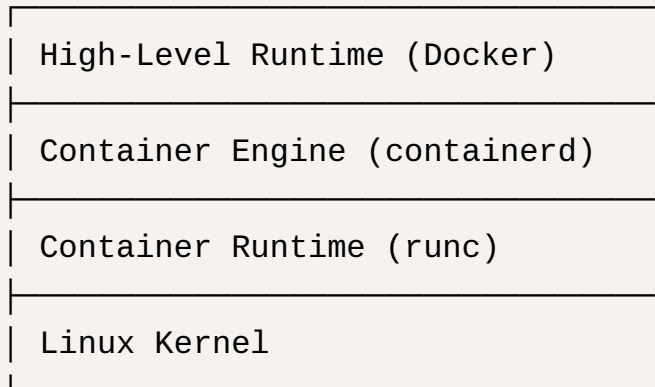
### 1. Docker Client

Docker CLI
- docker build
- docker pull
- docker run
- docker push

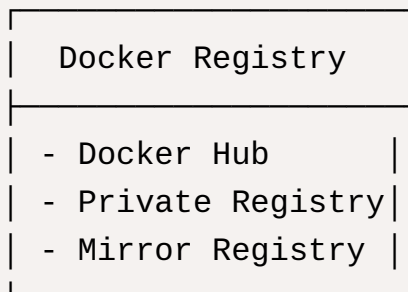
### 2. Docker Host



### 3. Container Runtime

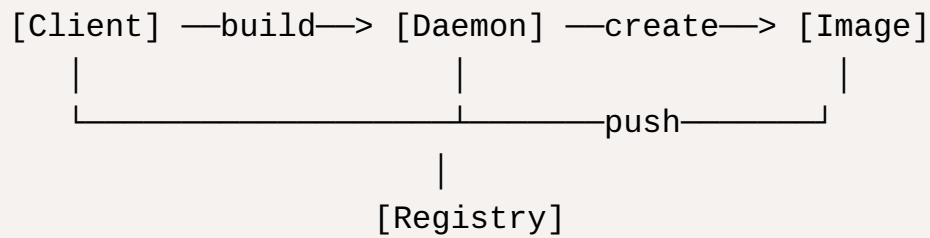


### 4. Registry

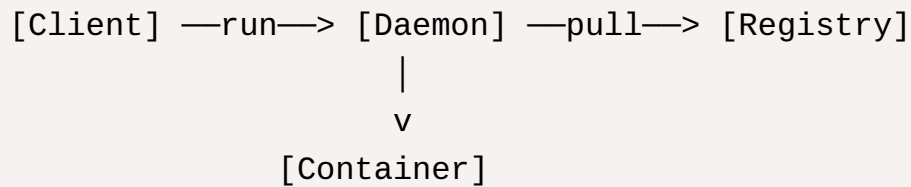


## Fluxo de Comunicação

### Build Flow



## Run Flow



## Componentes Detalhados

### Docker Daemon (dockerd)

- Gerencia objetos Docker
- Manipula API requests
- Gerencia containers
- Controla imagens

### containerd

- Gerenciamento de runtime
- Execução de containers
- Gerenciamento de imagens
- Interface com runc

### runc

- Criação de containers

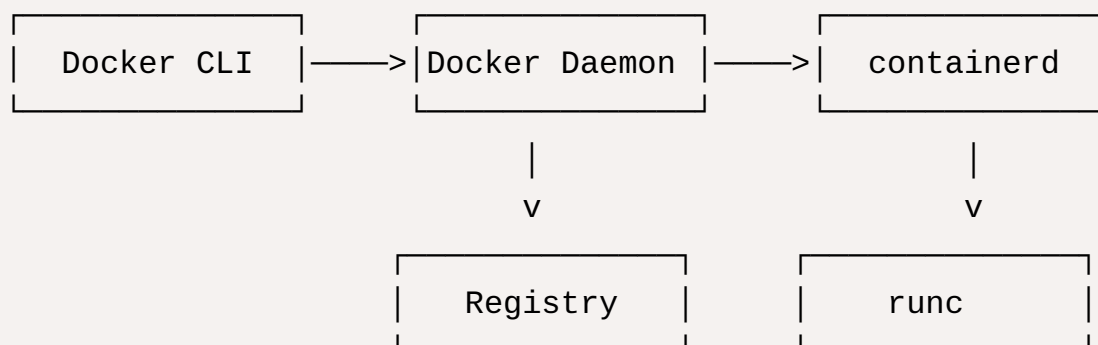
- Implementação OCI
- Interface com kernel
- Isolamento de recursos

## Waifu Tech Tips 💡

⚠️ **Architecture-chan diz:** "Cada componente tem seu papel! Como em um time de idol, todos precisam trabalhar juntos! 🎤"

⚠️ **Runtime-sama alerta:** "Nunca subestime o poder do container runtime - ele é o verdadeiro MVP! 🏆"

## Diagrama de Interação 📊



## Segurança e Isolamento 🔒

### Namespace Isolation

- PID Namespace
- Network Namespace
- Mount Namespace
- UTS Namespace

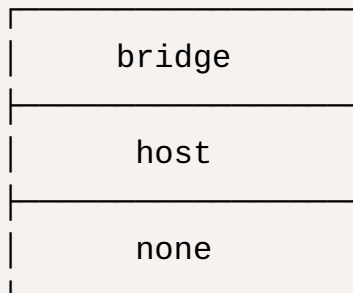
- IPC Namespace
- User Namespace

## Control Groups (cgroups)

- CPU
- Memory
- I/O
- Network
- Devices

## Networking Architecture

### Default Networks



### Custom Networks

- Bridge
- Overlay
- Macvlan
- IPvlan

## Storage Architecture

## Storage Drivers

- overlay2
- devicemapper
- btrfs
- zfs
- vfs

## Volume Types

Named Volumes
Bind Mounts
tmpfs Mounts

## Checkpoint

Você agora entende:

- [x] Componentes principais
- [x] Fluxo de comunicação
- [x] Container runtime
- [x] Networking
- [x] Storage
- [x] Segurança

## Próximos Passos

1. Container Runtime ([Container Runtime: O Motor dos Containers](#) 🚀)
2. Networking ([Container Networking](#))
3. Storage ([Container Storage: Crônicas dos Dados Perdidos](#) 💾)



"A arquitetura é como um bonsai - cada componente precisa ser cuidadosamente podado e mantido."

- Docker Architect-sensei, 2025

## Referências

- Docker Documentation (<https://docs.docker.com>)
- OCI Specification (<https://opencontainers.org>)
- containerd Documentation (<https://containerd.io>)



# CLI Basics: Dominando o Terminal Docker 🎮

```
|| DOCKER CLI COMMAND STRUCTURE ||
||                               ||
|| docker [options] command [arguments] ||
||                               ||
|| Examples: ||
|| docker run nginx ||
|| docker container ls ||
|| docker image build . ||
```

## Comandos Essenciais 🛠️

### Sistema e Informações

```
# Verificar versão
docker version

# Informações do sistema
docker info

# Status do daemon
docker system df

# Limpar recursos não utilizados
docker system prune -a
```

### Gerenciamento de Containers

```
# Listar containers
docker ps          # Ativos
```

```
docker ps -a          # Todos
docker ps -q          # Apenas IDs
docker ps --size      # Com tamanho

# Criar e executar
docker run nginx
docker run -d --name web nginx    # Detached mode
docker run -it ubuntu bash       # Interativo

# Operações básicas
docker start web
docker stop web
docker restart web
docker rm web
```

## Gerenciamento de Imagens

```
# Listar imagens
docker images
docker image ls

# Baixar imagem
docker pull nginx:latest

# Remover imagem
docker rmi nginx
docker image rm nginx

# Construir imagem
docker build -t app:1.0 .
```

## Flags Comuns

### Run Flags

```
# Portas
-p 8080:80          # Host:Container

# Volumes
-v /host:/container

# Variáveis de ambiente
-e MYSQL_ROOT_PASSWORD=secret

# Recursos
--memory 512m
--cpus 2

# Network
--network bridge
```

## Formato de Saída

```
# Formatação JSON
docker ps --format '{{json .}}'

# Tabela customizada
docker ps --format 'table {{.ID}}\t{{.Names}}\t{{.Status}}'
```

## Network Commands

```
# Listar redes
docker network ls

# Criar rede
docker network create mynet

# Conectar container
docker network connect mynet container1
```

```
# Inspeccionar rede
docker network inspect mynet
```

## Volume Commands

```
# Listar volumes
docker volume ls

# Criar volume
docker volume create mydata

# Inspeccionar volume
docker volume inspect mydata

# Remover volume
docker volume rm mydata
```

## Logs e Debug

```
# Ver logs
docker logs container1
docker logs -f container1    # Follow
docker logs --tail 100 container1

# Executar comando
docker exec -it container1 bash

# Inspeccionar container
docker inspect container1
```

## Dicas de Produtividade

### Aliases Úteis

```
# Adicione ao seu .bashrc ou .zshrc
alias d='docker'
alias dc='docker-compose'
alias dps='docker ps'
alias dim='docker images'
```

## One-Liners Poderosos

```
# Parar todos containers
docker stop $(docker ps -q)

# Remover containers parados
docker container prune

# Remover imagens não utilizadas
docker image prune -a

# Backup de container
docker commit container1 backup/container1
```

## Waifu CLI Tips 💡

⚠️ CLI-chan diz: "Sempre use `--help` quando precisar! É como ter um manual de bolso! 📖"

⚠️ Terminal-sama alerta: "Cuidado com `docker system prune -a`! Ele é tipo um delete sem recovery! 🗑️"

## Troubleshooting Common Issues 🔧

Erro	Causa Provável	Solução
Cannot connect to Docker daemon	Docker não está rodando	<code>sudo systemctl start docker</code>
Port is already allocated	Porta em uso	Use outra porta ou libere a atual
No space left on device	Disco cheio	<code>docker system prune</code>
Permission denied	Não está no grupo do docker	<code>sudo usermod -aG docker \$USER</code>

## Checkpoint

Você agora sabe:

- [x] Estrutura básica dos comandos
- [x] Gerenciar containers
- [x] Gerenciar imagens
- [x] Trabalhar com redes
- [x] Gerenciar volumes
- [x] Debug e logs
- [x] Aliases e produtividade

## Quick Reference Card



```


|| DOCKER CLI CHEAT SHEET ||
||
|| docker run      → Criar/Executar container ||

```

```
|| docker ps      → Listar containers ||
|| docker images  → Listar imagens    ||
|| docker build   → Construir imagem   ||
|| docker exec    → Executar comando   ||
|| docker logs    → Ver logs           ||
|| docker inspect → Inspeccionar objeto ||
|| docker prune   → Limpar recursos    ||
```

## Próximos Passos

1. Docker Commands ([Docker Commands: O Arsenal Completo](#) )
2. Docker Images ([Docker Images: Blueprint do Seu Container](#) )
- 3.
- 4.
- 5.

 **Sensei's Note:** "O CLI é sua espada no mundo Docker. Mantenha-a afiada e saiba quando usá-la!"

# Docker Commands: O Arsenal Completo 🚀

```
|| DOCKER COMMAND CATEGORIES ||  
|| Container | Image | Network | Volume | System | Compose ||
```

## Container Commands 📦

### Lifecycle Management

```
# Criar e Executar  
docker run [options] IMAGE  
-d, --detach          # Modo background  
-it                   # Modo interativo  
--name string         # Nome do container  
--rm                  # Remove após parar  
-p, --publish         # Mapear portas  
-v, --volume          # Montar volume  
  
# Gerenciamento  
docker start CONTAINER # Iniciar  
docker stop CONTAINER  # Parar  
docker restart CONTAINER # Reiniciar  
docker pause CONTAINER # Pausar  
docker unpause CONTAINER # Despausar  
docker rm CONTAINER    # Remover
```

### Monitoramento e Debug

```
# Status e Logs  
docker ps [options]    # Listar containers
```



```
docker logs CONTAINER      # Ver logs
docker stats CONTAINER     # Métricas em tempo real
docker top CONTAINER       # Processos
docker inspect CONTAINER   # Info detalhada

# Execução
docker exec [options] CONTAINER COMMAND
docker attach CONTAINER    # Conectar ao container
docker cp SRC_PATH CONTAINER:DEST_PATH
```

## Image Commands

### Gerenciamento de Imagens

```
# Básico
docker images              # Listar imagens
docker pull IMAGE          # Baixar imagem
docker push IMAGE          # Enviar imagem
docker rmi IMAGE           # Remover imagem

# Build
docker build [options] PATH
  -t, --tag                # Nomear imagem
  -f, --file                # Dockerfile alternativo
  --no-cache               # Ignorar cache

# Análise
docker history IMAGE       # Ver camadas
docker inspect IMAGE       # Info detalhada
docker save IMAGE          # Exportar
docker load                # Importar
```

## Network Commands

### Networking

```
# Gerenciamento
docker network create # Criar rede
docker network ls     # Listar redes
docker network rm     # Remover rede
docker network prune  # Limpar não usadas

# Conexões
docker network connect # Conectar container
docker network disconnect # Desconectar
docker network inspect # Inspecionar
```

## Volume Commands

### Storage

```
# Volumes
docker volume create # Criar volume
docker volume ls     # Listar volumes
docker volume rm     # Remover volume
docker volume prune  # Limpar não usados

# Inspeção
docker volume inspect # Info detalhada
```

## System Commands

### Manutenção

```
# Sistema
docker info      # Info do sistema
docker version   # Versão
docker events    # Eventos em tempo real

# Limpeza
docker system prune # Limpar tudo
```

```
docker system df      # Uso de disco
docker system info    # Info do sistema
```

## Compose Commands 🎵

### Orquestração

```
# Básico
docker-compose up      # Criar e iniciar
docker-compose down    # Parar e remover
docker-compose ps      # Listar serviços

# Lifecycle
docker-compose start   # Iniciar serviços
docker-compose stop    # Parar serviços
docker-compose restart # Reiniciar
```

## Command Flags Matrix 🎯

Flag	Uso	Exemplo
<code>-d</code>	Background	<code>docker run -d nginx</code>
<code>-p</code>	Portas	<code>docker run -p 80:80 nginx</code>
<code>-v</code>	Volumes	<code>docker run -v /host:/container nginx</code>
<code>-e</code>	Env vars	<code>docker run -e VAR=value nginx</code>
<code>--rm</code>	Auto-remove	<code>docker run --rm alpine</code>

## One-Liners Poderosos 💪

```
# Parar todos containers
docker stop $(docker ps -aq)

# Remover tudo
docker system prune -af --volumes

# Backup de container
docker commit container backup/container:v1

# Container efêmero
docker run --rm -it alpine sh
```

## Waifu Command Tips 🌸

⚠️ **Command-chan diz:** "Use `--help` com qualquer comando para ver todas as opções! 📖"

⚠️ **Docker-sama alerta:** "Cuidado com comandos destrutivos como `prune` e `rm -f!` 💣"

## Debug Arsenal 🔍

### Troubleshooting Commands

```
# Logs em tempo real
docker logs -f container

# Métricas live
docker stats container

# Processos do container
docker top container
```

```
# Dump completo  
docker inspect container > debug.json
```

## Quick Reference Card

DOCKER COMMANDS CHEAT SHEET		
Container	run, ps, start, stop, rm	
Image	build, pull, push, rmi	
Network	network create, connect, inspect	
Volume	volume create, ls, rm	
System	info, version, prune	
Compose	up, down, ps	

## Checkpoint

Você agora domina:

- [x] Comandos de container
- [x] Gerenciamento de imagens
- [x] Networking
- [x] Volumes
- [x] Manutenção do sistema
- [x] Docker Compose basics
- [x] Troubleshooting

## Próximos Passos

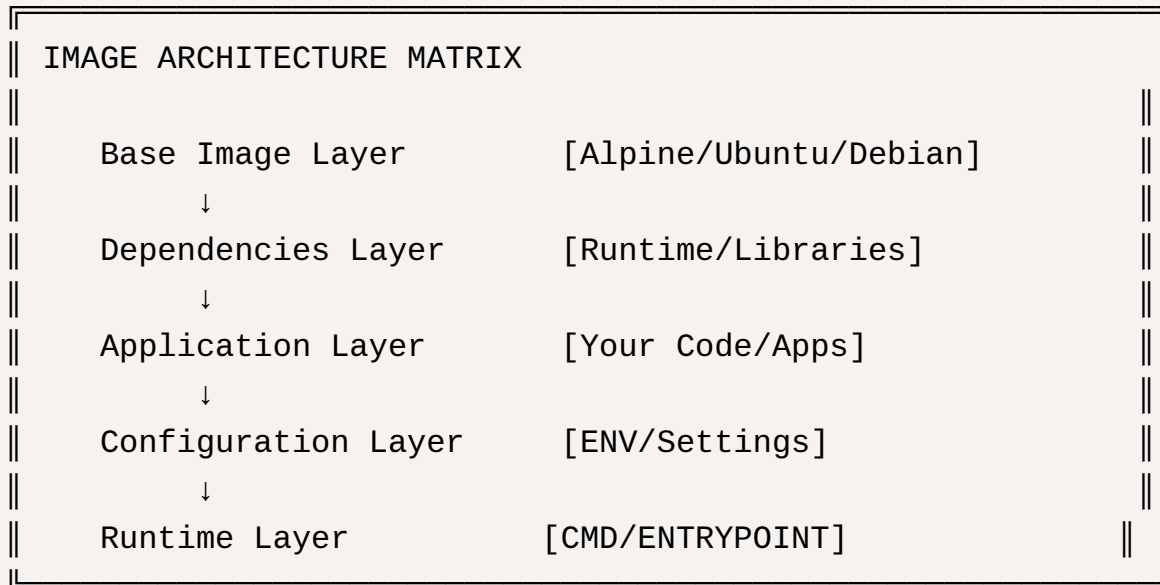
1. Container Operations ([Operações com Containers: O Manual do Operador](#) )

- 2.
- 3.
- 4.
- 5.



**Sensei's Note:** "Um verdadeiro mestre Docker conhece seus comandos como um samurai conhece sua espada!"

# Docker Images: Blueprint do Seu Container



## Fundamentos de Imagens

### Conceitos Básicos

- **Imagem:** Template read-only para containers
- **Container:** Instância executável de uma imagem
- **Layer:** Camada incremental de mudanças
- **Registry:** Repositório de imagens
- **Tag:** Identificador de versão
- **Digest:** Hash SHA256 único da imagem

### Tipos de Imagens

#### 1. Base Images

- Scratch
- Alpine (5MB)
- Debian Slim (80MB)
- Ubuntu (120MB)

## 2. Runtime Images

- node:alpine
- python:slim
- openjdk:slim

## 3. Application Images

- nginx:alpine
- postgres:alpine
- redis:alpine

# Anatomia Detalhada

## Sistema de Camadas

```
# Análise de camadas
docker history --no-trunc nginx:latest

# Metadata detalhada
docker inspect --format='{{.RootFS.Layers}}' nginx:latest

# Exportar imagem
docker save nginx:latest | tar -tvf -
```

## Estrutura Interna

```
# Exemplo de construção em camadas
FROM alpine:3.14
```



```
# Layer 1: Base OS

RUN apk add --no-cache python3
# Layer 2: Runtime

COPY ./app /app
# Layer 3: Application

ENV APP_ENV=production
# Layer 4: Configuration

CMD ["python3", "/app/main.py"]
# Layer 5: Runtime command
```

## Gerenciamento Avançado

### Image Operations

```
# Criar tag local
docker tag source:latest target:v1.0

# Salvar imagem como tar
docker save myapp:latest > myapp.tar

# Carregar imagem de tar
docker load < myapp.tar

# Filtrar imagens
docker images --filter "dangling=true"
docker images --filter "label=environment=prod"
```

### Batch Operations

```
# Remover todas imagens não utilizadas
docker image prune --all --force
```

```
# Remover por padrão
docker images -q "python*" | xargs docker rmi

# Limpar imagens antigas
docker images --format "{{.ID}}\t{{.CreatedAt}}" | sort -k 2 |
head -n 5 | cut -f1 | xargs docker rmi
```

## Sistema de Tags

### Convenções de Versionamento

```
# Tags específicas
registry.example.com/app:1.0.0
username/app:latest
custom/app:dev-build

# Multi-arquitetura
docker pull --platform linux/amd64 nginx
docker pull --platform linux/arm64 nginx
```

## Boas Práticas

### DO's

- Use tags específicas
- Minimize camadas
- Otimize cache
- Documente dependências
- Implemente multi-stage builds

### DON'Ts

- Evite `latest` em produção

- Não armazene secrets
- Não instale pacotes desnecessários
- Não ignore `.dockerignore`
- Não use imagens não oficiais sem verificação

## Image Operations Matrix

Operação	Comando	Uso Comum
Build	<code>docker build</code>	Criar nova imagem
Pull	<code>docker pull</code>	Baixar do registry
Push	<code>docker push</code>	Enviar para registry
Tag	<code>docker tag</code>	Criar alias/versão
Remove	<code>docker rmi</code>	Deletar imagem
Inspect	<code>docker inspect</code>	Ver metadata
Prune	<code>docker image prune</code>	Limpar não usadas

## Otimização de Imagens

### Redução de Tamanho

```
# Multi-stage build
FROM node:alpine AS builder
WORKDIR /app
COPY . .
RUN npm ci && npm run build
```

```
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

## Cache Optimization

```
# Ordem eficiente
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
```

## Registry Management

### Docker Hub

```
# Login
docker login [options] [SERVER]

# Logout
docker logout [SERVER]

# Pull de registry privado
docker pull private-registry.com/app:tag
```

## Advanced Security

### Image Signing

```
# Configurar DCT (Docker Content Trust)
export DOCKER_CONTENT_TRUST=1

# Gerar chaves
docker trust key generate mykey

# Assinar imagem
docker trust sign myregistry/myimage:tag
```

```
# Verificar assinatura
docker trust inspect --pretty myregistry/myimage:tag
```

## Security Scanning

```
# Trivy scan
trivy image python:3.9-alpine

# Snyc scan
snyc container test nginx:latest

# Anchore scan
anchore-cli image add nginx:latest
anchore-cli image wait nginx:latest
anchore-cli image vuln nginx:latest os
```

## Monitoring & Metrics

### Image Analytics

```
# Tamanho das camadas
docker history --no-trunc --format "{{.Size}}\t{{.CreatedBy}}"
nginx:latest

# Uso de disco
docker system df -v --format "
{{.Type}}\t{{.TotalCount}}\t{{.Size}}"

# Cache status
docker builder prune -f --filter until=24h
```

### Performance Tracking

```
# Build time analysis
time docker build --no-cache .
```

```
# Layer size tracking
docker image inspect myapp:latest -f '{{.RootFS.Layers}}'

# Pull time metrics
time docker pull large-image:latest
```

## Advanced Troubleshooting

### Debug Techniques

```
# Debug build
DOCKER_BUILDKIT=1 docker build --progress=plain .

# Layer inspection
docker save myimage:latest | tar -xC /tmp/image-layers

# Network issues
docker pull --verbose registry.example.com/image:tag
```

### Common Issues Matrix

Problema	Diagnóstico	Solução
Pull Timeout	<code>docker pull -v</code>	Verificar rede/proxy
Build Failure	<code>--no-cache</code>	Limpar cache/deps
Layer Issues	<code>docker history</code>	Otimizar Dockerfile
Space Issues	<code>docker system df</code>	Prune/cleanup

## Automation & CI/CD

### GitHub Actions

```

name: Docker Build

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v1

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Login to DockerHub
        uses: docker/login-action@v1
        with:
          username: ${ secrets.DOCKERHUB_USERNAME }
          password: ${ secrets.DOCKERHUB_TOKEN }

      - name: Build and push
        uses: docker/build-push-action@v2
        with:
          context: .
          platforms: linux/amd64,linux/arm64
          push: true
          tags: user/app:latest

```

## Automated Testing

```

# Test script
#!/bin/bash
set -e

# Build image

```

```
docker build -t test-image .

# Run container tests
docker run --rm test-image npm test

# Security scan
trivy image test-image

# Cleanup
docker rmi test-image
```

## Waifu Advanced Tips 🌟

⚠️ **Registry-chan diz:** "Mantenha seus registries organizados como uma coleção de mangás! 📖"

⚠️ **Build-sama alerta:** "Multi-stage builds são como transformações de magical girls - mais leves e poderosas! ✨"

ℹ️ **Cache-kun ensina:** "Cache é como um bento box - organize bem para reutilizar depois! 🍱"

## Quick Reference Pro 📋

ADVANCED IMAGE MANAGEMENT		
Multi-arch		docker buildx build --platform all
Sign		docker trust sign image:tag
Scan		trivy image nginx:latest
Analyze		docker history --no-trunc image
Debug		DOCKER_BUILDKIT=1 docker build .
Optimize		docker build --squash .



```
|| Monitor | docker events --filter type=image ||
```

## Advanced Checkpoint

Você agora domina:

- [x] Arquitetura interna de imagens
- [x] Multi-stage builds avançados
- [x] Registry privado e autenticação
- [x] Signing e scanning
- [x] Debug e troubleshooting
- [x] CI/CD integration
- [x] Performance optimization

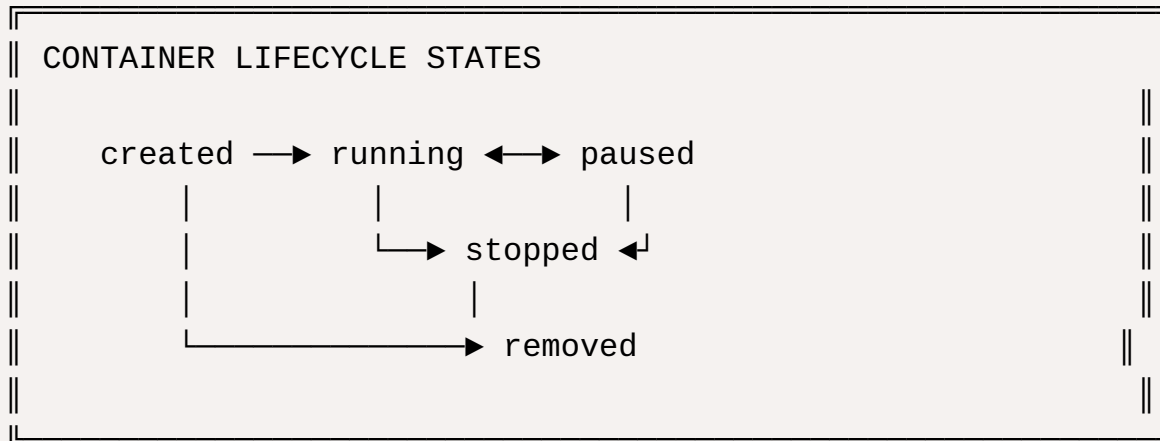
## Next Level Steps

- 1.
- 2.
- 3.
- 4.
- 5.



**Sensei's Final Note:** "Uma imagem Docker é como um kata perfeito - cada movimento tem propósito, precisão e eficiência! 🏆"

# Ciclo de Vida dos Containers: Do Nascimento à Transcendência



## Estados Fundamentais 🤖🔧

### 1. Created

```
# Criar container sem iniciar
docker create --name meu-container nginx

# Verificar estado
docker ps -a | grep meu-container

# Criar com configurações específicas
docker create \
  --name web-app \
  --memory 512m \
  --cpus 2 \
  --network bridge \
  nginx:latest
```

### 2. Running

```
# Iniciar container
docker start meu-container

# Criar e iniciar imediatamente
docker run -d --name outro-container nginx

# Executar com limites e variáveis
docker run -d \
  --name app \
  -p 8080:80 \
  -v /data:/app/data \
  -e NODE_ENV=production \
  --restart unless-stopped \
  node:alpine
```

### 3. Paused

```
# Pausar container
docker pause meu-container

# Despausar container
docker unpause meu-container

# Pausar múltiplos containers
docker pause $(docker ps -q)
```

### 4. Stopped

```
# Parar container gracefully
docker stop meu-container

# Parar container imediatamente
docker kill meu-container
```

```
# Parar com timeout personalizado
docker stop --time 30 meu-container
```

## 5. Removed

```
# Remover container parado
docker rm meu-container

# Forçar remoção
docker rm -f meu-container

# Remover todos containers parados
docker container prune
```

## Transições de Estado Avançadas

### Restart Policies

```
# No - não reinicia automaticamente
docker run --restart no nginx

# Always - sempre reinicia
docker run --restart always nginx

# On-failure - reinicia em caso de erro
docker run --restart on-failure:3 nginx

# Unless-stopped - reinicia exceto se parado manualmente
docker run --restart unless-stopped nginx
```

### Gerenciamento de Recursos

```
# Limitar memória e CPU
docker run \
  --memory 512m \
  --memory-swap 1g \
```

```
--cpus 2 \  
--cpu-shares 1024 \  
nginx  
  
# Atualizar limites em runtime  
docker update \  
  --memory 1g \  
  --cpus 4 \  
  container_id
```

## Monitoramento Avançado

### Eventos Detalhados

```
# Monitorar eventos específicos  
docker events \  
  --filter 'type=container' \  
  --filter 'event=start' \  
  --filter 'event=die' \  
  --filter 'event=stop' \  
  --format '{{.Time}} {{.Actor.ID}} {{.Action}}'  
  
# Stream de eventos em JSON  
docker events --format '{{json .}}'
```

### Métricas em Tempo Real

```
# Stats formatados  
docker stats --format "table  
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}"  
  
# Stats específicos  
docker stats \  
  --no-stream \  
  --format "{{.Container}}: CPU={{.CPUPerc}} MEM={{.MemPerc}}"
```

# Health Checks Avançados

## Configuração Detalhada

```
# Health check com múltiplas condições
docker run -d \
  --name web \
  --health-cmd="curl -f http://localhost/ || exit 1" \
  --health-interval=5s \
  --health-retries=3 \
  --health-timeout=2s \
  --health-start-period=30s \
  nginx

# Health check via shell script
docker run -d \
  --health-cmd='sh -c "curl -f http://localhost/ || exit 1"' \
  nginx
```

## Monitoramento de Saúde

```
# Verificar status detalhado
docker inspect \
  --format='{{json .State.Health}}' \
  container_id | jq

# Histórico de checks
docker inspect \
  --format='{{range .State.Health.Log}}{{.Start}}: {{.ExitCode}}\n{{end}}' \
  container_id
```

## Logging Avançado

### Configuração de Logs

```
# Configurar driver de log
docker run \
  --log-driver json-file \
  --log-opt max-size=10m \
  --log-opt max-file=3 \
  nginx

# Usar driver syslog
docker run \
  --log-driver syslog \
  --log-opt syslog-address=udp://logserver:514 \
  nginx
```

## Coleta de Logs

```
# Logs com timestamp
docker logs --timestamps container_id

# Logs desde específico timestamp
docker logs --since 2023-01-01T00:00:00 container_id

# Logs em formato JSON
docker logs --details container_id
```

## Automação e Orquestração

### Scripts de Manutenção Avançados

```
#!/bin/bash

# Reiniciar containers com base em uso de memória
docker stats --no-stream --format "{{.Container}} {{.MemPerc}}" | \
\
while read container memory; do
  if [ "${memory%.*}" -gt 90 ]; then
    echo "Reiniciando $container (Memória: $memory)"
```

```

    docker restart $container
fi
done

# Backup de containers
for container in $(docker ps -q); do
    name=$(docker inspect --format '{{.Name}}' $container | sed
's/\///')
    docker commit $container "${name}_backup"
    docker save "${name}_backup" > "${name}_$(date +%Y%m%d).tar"
done

```

## Integração com Docker Compose

```

version: '3.8'
services:
  web:
    image: nginx
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

```

## Troubleshooting Matrix

### Problemas Comuns



Problema	Sintoma	Solução	Comando
OOM Kill	Container morre	Aumentar memória	<code>docker update --memory 1g</code>
CPU Alto	Performance baixa	Limitar CPU	<code>docker update --cpus 2</code>
Zombie Processes	Processos órfãos	Reiniciar container	<code>docker restart</code>
Network Issue	Sem conectividade	Recriar rede	<code>docker network reconnect</code>

## Debug Commands

```
# Inspecionar processo init
docker top container_id

# Ver alterações no filesystem
docker diff container_id

# Debug de rede
docker exec container_id netstat -tulpn
```

## Lifecycle Hooks

### Pre-start Hook

```
docker run \
  --hook-spec='{"prestart": [{"path":
"/usr/local/bin/prestart.sh"}]}' \
  nginx
```

### Post-stop Hook

```
docker run \
  --hook-spec='{"poststop": [{"path":
"/usr/local/bin/cleanup.sh"}]}' \
  nginx
```

## Waifu Container Tips 💡

⚠️ **Lifecycle-chan diz:** "Containers são efêmeros! Trate-os como borboletas digitais! 🦋"

⚠️ **State-sama alerta:** "Sempre use stop antes de rm para um graceful shutdown! 🛑"




⚠️ **Debug-chan sugere:** "Logs são como pegadas digitais - siga-as para encontrar problemas! 🔍"


## Checkpoint Final ✅

Você agora domina:

- [x] Estados e transições dos containers
- [x] Políticas de restart
- [x] Monitoramento avançado
- [x] Health checks
- [x] Logging
- [x] Automação
- [x] Troubleshooting
- [x] Lifecycle hooks

## Próximos Passos

1. Container Operations ([Operações com Containers: O Manual do Operador](#) )
2. Container Runtime ([Container Runtime: O Motor dos Containers](#) )
3. Container Monitoring ([Container Monitoring: Os Olhos do Sistema](#) )
4. Container Networking ([Container Networking](#))
5. Container Security ([Segurança de Containers](#) )

 **Sensei's Note:** "Dominar o ciclo de vida dos containers é como se tornar um mestre Jedi - requer paciência, prática e sabedoria."

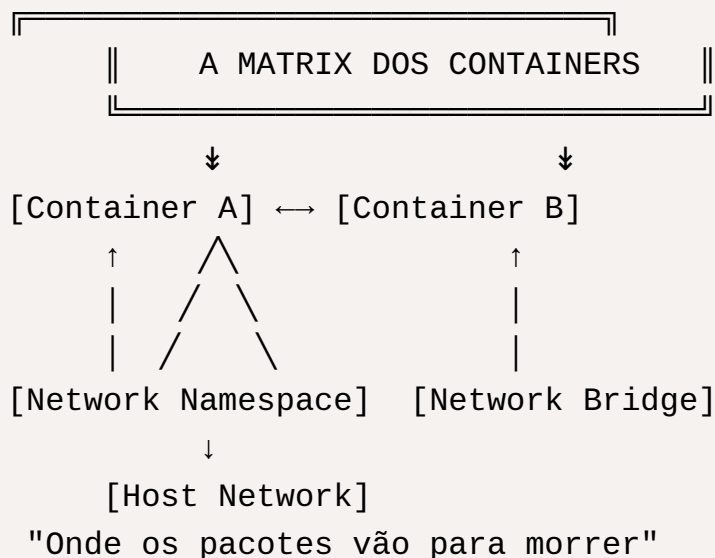
# Container Networking

⚠ "Em um mundo onde bits dançam através de cabos etéreos e pacotes viajam na velocidade da luz, eis que surge a mais intrigante das artes: a networking de containers." - Manifesto de um SRE Desiludido

## A Anatomia do Caos Digital 🌀

Ah, caro leitor! Permita-me guiá-lo através deste labirinto digital onde containers, qual habitantes de uma metrópole cyberpunk, comunicam-se através de pontes invisíveis e túneis etéreos.

### O Teatro das Sombras Digitais



Imagine, se puder, um teatro onde cada container é um ator mascarado, interpretando seu papel em um palco virtual. Cada um possui sua própria identidade (IP), seu próprio camarim (namespace), e suas próprias linhas de comunicação (veth pairs).

### Os Fantasmas da Máquina

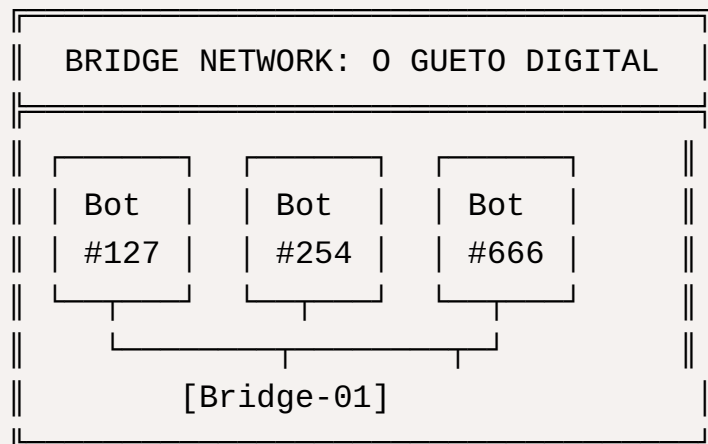
- **Network Namespaces:** Como apartamentos em um arranha-céu cyberpunk, cada container possui seu próprio espaço isolado. "Meu IP, minhas regras!"

- **Bridges Virtuais:** Pontes suspensas no vazio digital, conectando realidades paralelas. "A ponte para lugar nenhum, e ao mesmo tempo, para todo lugar."
- **veth pairs:** Cordões umbilicais digitais, nutrido bits e bytes. "Unidos na vida, unidos na morte."

## Os Drivers: Pilotos do Submundo Digital 🚗

### Bridge Network: O Submundo Comum

Como becos escuros de uma cidade neon, a bridge network é onde a maioria dos containers vive - não é bonito, mas é funcional.



### Host Network: A Liberdade Perigosa

Ah, a rede do host! Como um mergulho sem paraquedas em Neo-Tokyo. Performance máxima, segurança mínima. Viva perigosamente!

```
# Para os corajosos (ou loucos)
docker run --network host nginx
# "Quem precisa de isolamento quando se tem velocidade?" - DevOps
antes do incidente
```

### Overlay Network: A Teia da Ilusão

Multi-host networking, ou como gosto de chamar: "A arte de fazer containers acreditarem que estão na mesma rede quando, na verdade, estão separados por um abismo digital."

## Configuração Multi-host

```
# Inicializar swarm
docker swarm init

# Criar rede overlay
docker network create \
  --driver overlay \
  --attachable \
  --subnet=10.0.9.0/24 \
  --gateway=10.0.9.1 \
  overlay_net

# Criptografia
docker network create \
  --driver overlay \
  --opt encrypted=true \
  secure_overlay
```

## Macvlan

- Interfaces MAC dedicadas
- Performance próxima ao bare metal
- Integração com redes físicas

## Modos de Operação

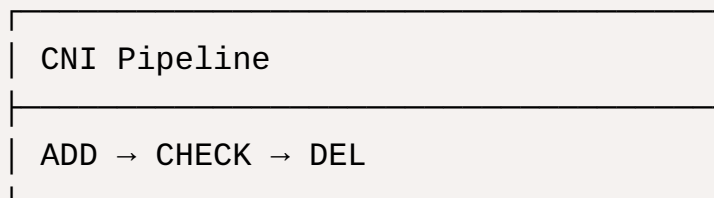
```
# Modo Bridge
docker network create -d macvlan \
  --subnet=192.168.32.0/24 \
  --gateway=192.168.32.1 \
  -o parent=eth0 \
  -o macvlan_mode=bridge \
  macvlan_bridge

# Modo 802.1q trunk
```

```
docker network create -d macvlan \  
  --subnet=192.168.32.0/24 \  
  --gateway=192.168.32.1 \  
  -o parent=eth0.10 \  
  macvlan_trunk
```

## Container Network Interface (CNI)

### Arquitetura CNI



### Plugins CNI

#### Básicos

- bridge
- host-local
- loopback
- dhcp

#### Metaplugins

- flannel
- tuning
- portmap
- bandwidth

#### Plugins Avançados

- Calico
  - BGP networking
  - Network policies
  - IPAM
- Cilium
  - eBPF-based networking
  - L7 policies
  - Observability
- Weave
  - Encrypted networking
  - Service discovery
  - Fast datapath

## Configuração de Rede

### Port Mapping

```
# Exposição básica
docker run -p 8080:80 nginx

# UDP ports
docker run -p 53:53/udp dns-server

# Múltiplos ports
docker run \
  -p 80:80 \
  -p 443:443 \
  -p 8080:8080 \
  web-server
```



```
# Range de portas
docker run -p 8080-8090:80-90 nginx
```

## DNS e Service Discovery

### Configuração DNS

```
# /etc/docker/daemon.json
{
  "dns": ["8.8.8.8", "8.8.4.4"],
  "dns-search": ["example.com"],
  "dns-opts": ["ndots:2", "timeout:2"]
}
```

### Docker Compose

```
version: '3.8'
services:
  web:
    networks:
      frontend:
        aliases:
          - web.local
      backend:
        aliases:
          - web.internal
  api:
    networks:
      backend:
        ipv4_address: 172.16.238.10

networks:
  frontend:
    driver: bridge
    enable_ipv6: true
  ipam:
    driver: default
```

```
    config:
      - subnet: 172.16.238.0/24
      - subnet: "2001:db8:1::/64"
  backend:
    internal: true
    driver: bridge
```

## Segurança de Rede

### Network Policies

#### Isolamento Básico

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```

#### Políticas Avançadas

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      role: api
  policyTypes:
    - Ingress
    - Egress
  ingress:
```

```

- from:
  - namespaceSelector:
      matchLabels:
        purpose: production
  ports:
  - protocol: TCP
    port: 8080
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/24
      except:
      - 10.0.0.1/32
  ports:
  - protocol: TCP
    port: 5432

```

## Isolamento

### Técnicas de Isolamento

- Network namespaces
- Bridge isolation
- VLAN tagging
- Security groups

### Firewall Rules

```

# Bloquear acesso externo
iptables -I DOCKER-USER -i ext_if ! -s 192.168.1.0/24 -j DROP

# Permitir apenas portas específicas
iptables -I DOCKER-USER -i ext_if -p tcp --dport 80 -j ACCEPT
iptables -I DOCKER-USER -i ext_if -p tcp --dport 443 -j ACCEPT

```

# Troubleshooting

## Ferramentas de Diagnóstico

### Network Inspection

```
# Detalhes da rede
docker network inspect bridge

# Lista de endpoints
docker network inspect -f '{{range .Containers}}{{.Name}}
{{.IPv4Address}}{{end}}' bridge

# Estatísticas de rede
docker stats --no-stream --format "table
{{.Container}}\t{{.NetIO}}"
```

### Debugging Tools

```
# tcpdump
docker run --net=container:target \
  --privileged \
  nicolaka/netshoot \
  tcpdump -i any port 80

# netstat
docker exec container netstat -tulpn

# iptables
docker run --net=host --privileged \
  nicolaka/netshoot \
  iptables -nvL
```

## Problemas Comuns

### Matriz de Troubleshooting

Problema	Diagnóstico	Solução	Comando
DNS	nslookup	Verificar resolv.conf	<code>docker exec container cat /etc/resolv.conf</code>
Conectividade	ping	Verificar firewall	<code>iptables -nvL</code>
Portas	netstat	Verificar binding	<code>netstat -tulpn</code>
Performance	iperf	Testar bandwidth	<code>iperf -c target</code>

## Monitoramento

### Métricas de Rede

#### Básicas

```
# Estatísticas básicas
docker stats --format "table {{.Name}}\t{{.NetIO}}"

# IO detalhado
docker stats --no-stream --format "table
{{.Container}}\t{{.NetIO}}\t{{.BlockIO}}"
```

#### Avançadas

```
# Monitoramento com cAdvisor
docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --publish=8080:8080 \
```

```
--detach=true \  
--name=cadvisor \  
google/cadvisor:latest
```

## Logging

### Configuração de Log

```
{  
  "log-driver": "json-file",  
  "log-opts": {  
    "max-size": "10m",  
    "max-file": "3"  
  }  
}
```

### Coleta de Logs

```
# Logs de rede  
docker logs --tail 100 -f container  
  
# Eventos de rede  
docker events --filter 'type=network'
```

## Referências

### Documentação

- Docker Networking Overview (<https://docs.docker.com/network/>)
- CNI Specification  
(<https://github.com/containernetworking/cni/blob/master/SPEC.md>)
- Libnetwork Design  
(<https://github.com/docker/libnetwork/blob/master/docs/design.md>)

### Ferramentas

- Netshoot (<https://github.com/nicolaka/netshoot>)
- Network Debug Tools (<https://github.com/docker/docker-diagnose>)
- Network Plugins (<https://github.com/container networking/plugins>)

## Tutoriais Avançados

- Multi-host Networking (<https://docs.docker.com/network/overlay/>)
- Network Security (<https://docs.docker.com/network/security/>)
- Custom Network Plugins ([https://docs.docker.com/engine/extend/plugins\\_network/](https://docs.docker.com/engine/extend/plugins_network/))

## Recursos Adicionais

- Docker Network Troubleshooting Guide  
(<https://success.docker.com/article/troubleshooting-container-networking>)
- Container Network Interface Specification  
(<https://github.com/container networking/cni/blob/master/SPEC.md>)
- Docker Network Architecture  
(<https://docs.docker.com/engine/userguide/networking/dockernetworks/>)

# Container Storage: Crônicas dos Dados Perdidos

**⚠** "Em um mundo onde dados são mais valiosos que créditos digitais, a arte do armazenamento em containers é como dança macabra entre persistência e efemeridade." - Memórias de um DBA Cybernético

## O Teatro das Sombras Binárias

STORAGE MATRIX: O PARADOXO DA PERSISTÊNCIA

"Todos os dados são efêmeros,  
até que precisemos deles novamente"

## A Anatomia do Vazio Digital

Ah, caro aventureiro dos bytes! Permita-me guiá-lo através deste labirinto de dados, onde containers nascem e morrem como sonhos em uma noite de neón, levando consigo seus preciosos dados... ou será que não?

Container Life & Death

Efêmero

vs

Persistente

"Memento Mori Digital"

## Os Três Cavaleiros do Armazenamento



## 1. Volumes: O Imortal Digital

Como vampiros em um romance gótico-tech, os volumes Docker persistem além da morte de seus containers.

```
# Invocando o ritual do volume
docker volume create dados-imortais
docker run -v dados-imortais:/data nginx
# "Nem mesmo um 'docker rm -f' pode matar estes dados"
```

## 2. Bind Mounts: As Correntes Digitais

Como fantasmas acorrentados a uma mansão antiga, bind mounts ligam seu container ao mundo mortal do host.

```
# O ritual de ligação
docker run -v /home/user/dados:/app/dados nginx
# "Presos para sempre ao sistema host, como uma maldição digital"
```

## 3. tmpfs: O Sussurro Efêmero

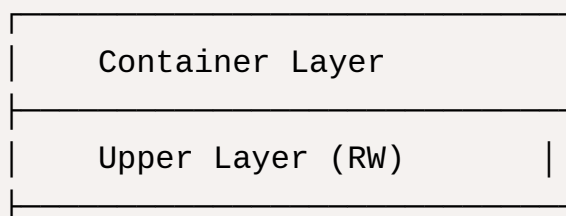
Como pensamentos em uma mente cibernética, tmpfs existe apenas na memória volátil.

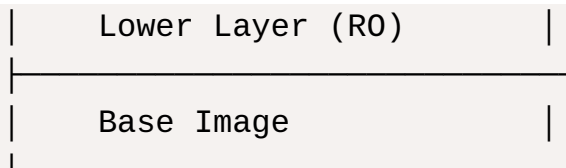
```
# A dança do esquecimento
docker run --tmpfs /app/temp nginx
# "Aqui jazem dados temporários, que viveram rápido e morreram jovens"
```

## O Overlay2: Dança das Camadas

### A Valsa das Camadas

Como um baile de máscaras digital, onde cada camada esconde segredos:





"Cada camada, uma história não contada"

## Storage Drivers: Os Necromantes ♂

### O Panteão dos Drivers

- **overlay2**: O príncipe herdeiro, elegante e eficiente
- **devicemapper**: O velho rei, complexo mas poderoso
- **btrfs**: O místico incompreendido
- **zfs**: O mago das pools
- **aufs**: O ancestral aposentado

DRIVER SELECTION MATRIX

"Escolha seu veneno digital"

## Waifu Storage Tips

**⚠ Volume-chan avisa:** "Seus dados são como primeiros amores - guarde-os em volumes, ou chore sua perda eternamente! uwu"

**⚠ Backup-sama declara:** "Na matrix dos containers, não existem segundas chances... a menos que você tenha um backup. Ara ara~"

## Rituais de Performance 🏃♂️

### O Cache Dance

```
# O ritual do cache
--volume-opt size=20G
--volume-opt o=size=20G
# "Dança mais rápido, meu precioso cache!"
```

### A Arte da Otimização

```
volumes:
  data:
    driver: local
    driver_opts:
      type: 'none'
      o: 'bind'
      device: '/data/mysql'
# "Configuração é poesia em YAML"
```

## Troubleshooting: Necropsia Digital 🔍

### Sinais do Além

```
# Investigando anomalias
docker volume ls
docker volume inspect meu-volume
# "Cada erro conta uma história macabra"
```

## Exercícios Para os Destemidos 🎮

1. **A Cripta:** Configure um volume persistente sem perder dados
2. **O Espelho:** Sincronize dados entre host e container
3. **O Vazio:** Gerencie tmpfs sem crashar a memória

## Palavras Finais

Como diria o enigmático DevOps Poe: "Em cada container mora um dado, esperando para ser perdido ou salvo. A escolha, meu caro operador, é sua."



"E assim terminamos nossa jornada pelos calabouços do armazenamento. Lembre-se: na próxima vez que seus dados desaparecerem com um container, não foi um bug - foi apenas o karma digital cobrando seu preço." - Chronicles of a Storage Engineer, 2084