Docker — Treinamento Corporativo

Comandos, fundamentos, práticas de produção e laboratórios guiados

Hands-on • CLI • Dockerfile • Compose • Troubleshooting

Duração sugerida: 4–8h (adaptável)

Agenda do treinamento

Estrutura modular — pare aqui para encaixar pausas e dinâmicas

Módulos

- M1: Fundamentos (imagens, containers, CLI)
- M2: Execução e debug (logs, exec, inspect)
- M3: Persistência e rede (volumes, networks)
- M4: Dockerfile (build, layers, boas práticas)
- M5: Docker Compose (multi-serviço)
- M6: Registro/Distribuição (tag, push, pull)
- M7: Segurança & produção (scan, least privilege)
- M8: Troubleshooting & limpeza (prune, diagnóstico)

Dinâmica

- Blocos curtos de teoria (10–15 min)
- Demonstração ao vivo (5–10 min)
- Laboratório guiado (20–40 min)
- Knowledge check (3–5 perguntas)
- Checklist de entrega (o que deve funcionar)
- Tempo para Q&A e incidentes reais

Objetivos de aprendizagem

O que a turma deve conseguir fazer ao final

Ao final, você será capaz de:

- Explicar imagens vs containers e o ciclo de vida de execução
- Executar, inspecionar, debugar e coletar logs de containers
- Persistir dados com volumes e conectar serviços com networks
- Criar Dockerfiles otimizados (cache, multi-stage, segurança)
- Orquestrar ambientes locais com Docker Compose
- Versionar e publicar imagens em registries com boas práticas
- Aplicar hardening básico e padrões de produção
- Diagnosticar problemas comuns e limpar recursos com segurança

Pré-requisitos e ambiente

Padronize para reduzir “funciona na minha máquina”

Recomendado

- Docker Desktop (Win/Mac) ou Docker Engine (Linux)
- WSL2 no Windows (se aplicável)
- Git + editor (VS Code recomendado)
- Portas livres: 3000, 5432, 6379 (para labs)
- Acesso a um registry (Docker Hub / GHCR / registry interno)

Validação rápida

```
docker version

docker info

docker run --rm hello-world
```

Se “hello-world” executar, o ambiente está OK.

Política corporativa (exemplo)

- Imagens base aprovadas (ex.: debian-slim, alpine, distroless)
- Repositórios privados e autenticação (SSO/Token)
- Proibição de rodar containers privilegiados sem justificativa
- Padronização de labels, ports e healthchecks
- Logs padronizados (stdout/stderr) e observabilidade

M1

M1 — Fundamentos

O que é Docker, imagem, container e comandos essenciais

Modelo mental

Imagem é o “template”; container é a instância em execução

Imagem

- Camadas (layers) imutáveis
- Construída via Dockerfile (docker build)
- Identificada por: nome:tag e digest
- Armazenada localmente e/ou em registry

Container

- Processo isolado (namespaces/cgroups)
- Filesystem “copy-on-write” sobre a imagem
- Estado: created/running/stopped
- Descartável — estado vai para volumes/bancos

Hello (fundamentos)

```
docker pull nginx:alpine

docker run --name web -d -p 8080:80 nginx:alpine

docker ps

docker stop web
docker rm web
```

Demonstração: pull → run → ps → stop → rm.

Comandos base (CLI)

Mapa rápido do dia a dia

Operações comuns

- docker version / info
- docker images / image ls
- docker ps (-a) / container ls
- docker run / start / stop / restart
- docker rm / rmi
- docker logs / exec / inspect
- docker network ls / volume ls
- docker system df / prune

Exemplo: run completo

```
docker run --name app   -e NODE_ENV=production   -p 3000:3000   --restart
unless-stopped   -d node:20-alpine   node -e
"require('http').createServer((_,r)=>r.end('ok')).listen(3000)"
```

Use --rm em labs para evitar lixo (quando aplicável).

LAB 1 — Primeiros passos (10–15 min)

Entrega: Nginx respondendo e container gerenciado corretamente

Tarefas

- 1) Faça pull do nginx:alpine
- 2) Rode um container mapeando 8080→80
- 3) Liste containers e identifique o ID
- 4) Pare, inicie e remova o container
- 5) Refaça usando --rm e compare o comportamento

Comandos sugeridos

```
docker pull nginx:alpine

docker run --name web -d -p 8080:80 nginx:alpine

docker ps

docker stop web

docker start web

docker rm -f web

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

Use o navegador/curl para validar http://localhost:8080

Checklist de entrega

- Porta 8080 responde com página padrão do Nginx
- Você consegue identificar nome, ID e estado do container
- Você removeu o container e validou que ele não aparece no docker ps -a

Knowledge check — M1

Perguntas rápidas (discussão em duplas)

Responda (sem consultar):

- Qual a diferença entre docker run e docker start?
- O que muda ao usar --rm?
- Como você expõe uma porta do container para o host?
- Quando usar -it?
- Por que containers devem ser “descartáveis”?

M2

M2 — Execução e Debug

Logs, exec, inspect, health e troubleshooting rápido

Logs e processos

Diagnóstico básico sem “entrar” no container

Logs e follow

```
docker logs <container>

docker logs -f --tail 200 <container>

docker top <container>

docker stats --no-stream
```

Dica: padronize logs em stdout/stderr — sem gravar em arquivo local.

Boas práticas

- Logs: use níveis (info/warn/error)
- Inclua request-id/correlation-id
- Evite logs com dados sensíveis
- Rotacione/colecione fora do container

Comandos úteis

- docker events (auditoria rápida)
- docker diff (mudanças no fs)
- docker cp (copiar arquivos)
- docker port (mapeamentos)

docker exec vs docker attach

Quando usar cada um

docker exec

- Roda um comando “novo” dentro do container
- Não interfere no PID 1 da aplicação
- Ideal para debug (sh, cat, curl)
- Ex.: docker exec -it app sh

docker attach

- Conecta ao STDIN/STDOUT do processo principal
- Pode travar se o app não espera input
- Desconectar pode parar o container (depende do app)

Exemplo prático

```
docker run --name app -d nginx:alpine

docker exec -it app sh
# dentro: apk add --no-cache curl
# dentro: curl -I http://localhost
exit

docker inspect app --format '{{json .State}}'
```

Use inspect para ver status, health, mounts e redes.

LAB 2 — Debug guiado (20–25 min)

Entrega: diagnosticar e corrigir um container “quebrado”

Cenário

- Você recebeu um container que “não sobe” ou sai imediatamente.
- Objetivo: identificar o erro (logs/inspect) e corrigir o comando.
- Regra: primeiro investigue fora (logs/ps/inspect). Só depois use exec.

Reprodução do problema

```
docker run --name broken -d nginx:alpine nginx -g "daemon off;" -c /nope.conf

docker ps -a

docker logs broken

docker inspect broken --format '{{.State.ExitCode}}'
```

ExitCode != 0 → container parou. Corrija o comando e rode novamente.

Checklist

- Você coletou logs e exit code
- Você explicou por que o processo terminou
- Você corrigiu o comando e deixou o container “running”

M3

M3 — Volumes & Networks

Persistência, bind mounts, redes bridge e DNS interno

Persistência

Containers são descartáveis — dados não

Tipos

- Volumes (gerenciados pelo Docker) — recomendado
- Bind mounts (pasta do host) — ótimo para dev
- tmpfs (memória) — dados temporários

Volume: criar/usar

```
docker volume create pgdata

docker run --name db -d   -e POSTGRES_PASSWORD=postgres   -v
pgdata:/var/lib/postgresql/data   -p 5432:5432   postgres:16-alpine

# Remova o container e recrie: os dados persistem
```

Use “-v nome:/caminho” para volumes nomeados.

Bind mount: dev rápido

```
docker run --name web -d -p 8080:80   -v "$(pwd)/site:/usr/share/nginx/html:ro"   nginx:alpine
```

Ideal para hot-reload (dev). Em produção, prefira imagem imutável.

Redes

Bridge por padrão + DNS interno por nome do container

Conceitos

- network bridge: padrão local
- containers na mesma rede resolvem nomes (DNS)
- publish (-p) expõe porta para o host
- expose é metadado (Compose usa)

Criar rede e conectar

```
docker network create appnet

docker run -d --name redis --network appnet redis:7-alpine

docker run -it --rm --network appnet alpine:3.20 sh
# dentro: apk add --no-cache redis
# dentro: redis-cli -h redis PING
```

Sem IP fixo — use o nome “redis”.

Boas práticas corporativas

- Evite depender de “localhost” entre containers
- Use uma rede por stack/sistema
- Exponha portas somente quando necessário
- Padronize variáveis: DB_HOST=db, REDIS_HOST=redis

LAB 3 — Banco + app em rede (30–40 min)

Entrega: app acessa Postgres por DNS da rede

Tarefas

- 1) Crie network appnet
- 2) Suba Postgres na rede (sem expor para host)
- 3) Suba um container “client” e conecte via psql
- 4) (Opcional) Exponha 5432 e conecte via ferramenta local

Roteiro

```
docker network create appnet

docker run -d --name db --network appnet   -e POSTGRES_PASSWORD=postgres postgres:16-alpine

docker run -it --rm --network appnet postgres:16-alpine   psql -h db -U postgres -c "SELECT now();"
```

Se falhar: verifique logs do db e variáveis.

Checklist

- psql conectou usando host=db (nome do container)
- Você sabe explicar por que não precisa de IP fixo
- Você entende a diferença entre “na rede” vs “expor porta”

M4

M4 — Dockerfile & Build

Build, cache, camadas, multi-stage e hardening básico

Dockerfile: estrutura

Otimização de cache e previsibilidade

Instruções comuns

- FROM, WORKDIR, COPY, RUN
- ENV, ARG
- EXPOSE (metadado)
- USER (não-root)
- CMD vs ENTRYPOINT
- HEALTHCHECK

Build e tag

```
docker build -t minhaapp:1.0 .

docker image ls

docker run --rm -p 3000:3000 minhaapp:1.0
```

Sempre fixe versões (evite latest em produção).

Dockerfile exemplo (Node)

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
EXPOSE 3000
CMD ["node","server.js"]
```

Coloque COPY de dependências antes para aproveitar cache.

Multi-stage build

Imagem final menor e mais segura

Exemplo (Node + build)

```
# stage 1 — build
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# stage 2 — runtime
FROM node:20-alpine
WORKDIR /app
ENV NODE_ENV=production
COPY package*.json ./
RUN npm ci --omit=dev
COPY --from=build /app/dist ./dist
USER node
EXPOSE 3000
CMD ["node","dist/server.js"]
```

Resultado: sem ferramentas de build na imagem final.

Boas práticas corporativas

- Use .dockerignore para reduzir contexto
- Fixe versões (FROM node:20.11-alpine etc.)
- Não rode como root, sempre que possível

LAB 4 — Build + run (30–45 min)

Entrega: imagem pequena + container rodando + healthcheck

Tarefas

- 1) Criar um Dockerfile (app simples)
- 2) Buildar com tag v1
- 3) Rodar e validar endpoint
- 4) Implementar multi-stage e comparar tamanho (docker images)
- 5) Adicionar HEALTHCHECK (básico) e validar no inspect

Sugestão (healthcheck)

```
HEALTHCHECK --interval=30s --timeout=3s --retries=3   CMD wget -qO-
http://localhost:3000/health || exit 1
```

Se sua base não tem wget/curl, instale no build ou ajuste a imagem.

Validação

```
docker build -t app:v1 .

docker run -d --name app -p 3000:3000 app:v1

docker ps

docker inspect app --format '{{json .State.Health}}'
```

Atenção: health pode levar alguns segundos para mudar.

M5

M5 — Docker Compose

Ambientes multi-serviço, variáveis, depends_on e profiles

Por que Compose?

Stack local reproduzível para dev/test

Vantagens

- Define serviços, redes e volumes em YAML
- Subida/derrubada com 1 comando
- Padroniza variáveis e portas
- Facilita onboarding e troubleshooting

Comandos

```
docker compose up -d

docker compose logs -f

docker compose ps

docker compose down -v
```

Use “down -v” com cuidado — remove volumes (dados).

compose.yaml (exemplo completo)

App + Postgres + Redis + redes/volumes

Arquivo: compose.yaml

```yaml
services:
  api:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DB_HOST=db
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - REDIS_HOST=redis
    depends_on:
      - db
      - redis

  db:
    image: postgres:16-alpine
    environment:
      - POSTGRES_PASSWORD=postgres
    volumes:
      - pgdata:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine

volumes:
  pgdata:
```

Dica: se quiser “esperar” DB pronto, use healthchecks + condition (quando aplicável).

LAB 5 — Stack com Compose (40–60 min)

Entrega: subir stack e validar conectividade

Tarefas

- 1) Criar compose.yaml com api + db + redis
- 2) Subir com docker compose up -d
- 3) Validar logs e status
- 4) Entrar no container api e testar resolução DNS
- 5) Derrubar sem apagar volumes (down) e com apagar volumes (down -v)

Comandos de validação

```
docker compose up -d

docker compose ps

docker compose logs -f api

docker compose exec api sh
# dentro: nslookup db
# dentro: nslookup redis
exit

docker compose down
```

Em produção, Compose é útil; mas orquestração maior costuma ser Kubernetes/Swarm.

Checklist

- 3 serviços “running”
- api resolve db/redis por nome
- Você sabe explicar o impacto de “down -v”

M6

M6 — Registry

Tagging, push/pull, versionamento e boas práticas

Nomes, tags e digests

Estratégia de versionamento evita “quebrar” produção

Regras práticas

- Use tags semânticas: 1.2.3, 1.2, 1, latest (apenas dev)
- Trave por digest em produção quando necessário
- Inclua labels: commit sha, build date, owner
- Nunca use senha em linha de comando (use stdin/token)

Tag + push

```
docker login

docker tag app:v1 registry.exemplo.local/app:1.0.0

docker push registry.exemplo.local/app:1.0.0

docker pull registry.exemplo.local/app:1.0.0
```

Ajuste o registry conforme sua empresa (Docker Hub/GHCR/ECR/GCR/ACR).

Labels (exemplo)

```
LABEL org.opencontainers.image.source=$REPO_URL org.opencontainers.image.revision=$GIT_SHA org.opencontainers.image.created=$BUILD_DATE
```

Esses metadados ajudam auditoria e rastreabilidade.

LAB 6 — Publicar imagem (20–30 min)

Entrega: imagem publicada e baixada em outra máquina/perfil

Tarefas

- 1) Criar uma tag com versão (ex.: 1.0.0)
- 2) Fazer login no registry
- 3) Push da imagem
- 4) Remover imagem local (rmi) e fazer pull
- 5) Rodar novamente e validar

Roteiro

```
docker login

docker tag app:v1 <seu-usuario>/app:1.0.0

docker push <seu-usuario>/app:1.0.0

docker rmi <seu-usuario>/app:1.0.0

docker pull <seu-usuario>/app:1.0.0
```

Se estiver em registry privado, valide DNS/VPN e permissões.

Checklist

- A tag 1.0.0 existe no registry
- Você consegue pull e run sem build local
- Você entende diferença entre tag e digest

M7

M7 — Segurança & Produção

Hardening, mínimos privilégios, scanning e políticas

Hardening básico

Padrões que reduzem risco sem complicar

Checklist

- Rodar como usuário não-root (USER)
- Evitar --privileged e caps desnecessárias
- Filesystem somente leitura quando possível (--read-only)
- Definir limites: --memory, --cpus
- Não embutir segredos na imagem
- Fixar base image e atualizar CVEs

Run com hardening (ex.)

```
docker run -d --name api   --read-only   --tmpfs /tmp   --cap-drop ALL   --security-opt
no-new-privileges   --memory 512m --cpus 1   -p 3000:3000 app:1.0.0
```

Nem todo app funciona read-only — avalie e ajuste paths.

Observabilidade em produção

- Logs em stdout/stderr
- Healthcheck para readiness/liveness (no orquestrador)
- Métricas (Prometheus/OpenTelemetry)
- Tracing e correlação por request-id

Scanning e SBOM (visão geral)

Integração em CI/CD para bloquear CVEs críticas

Práticas recomendadas

- Scan de imagem no pipeline (antes do deploy)
- Geração de SBOM (dependências) para auditoria
- Políticas: bloquear CVEs críticas e high
- Assinatura/verificação de imagens (quando adotado)

Exemplos (depende do stack)

```
# exemplo conceitual (pode variar)
# trivy image app:1.0.0
# syft app:1.0.0 -o spdx-json
# cosign sign ...
```

A ferramenta exata varia por empresa (Trivy/Grype/Snyk etc.).

Atenção

- Scanning não substitui patching e atualização de base images
- Evite imagens gigantes (reduz superfície)
- Use reprodutibilidade: lockfiles + versões fixas

M8

M8 — Troubleshooting & Limpeza

Diagnóstico rápido, problemas comuns e prune seguro

Problemas comuns

Sintoma → hipótese → comando

Sintomas frequentes

- Porta já em uso (bind: address already in use)
- Container sai imediatamente (ExitCode != 0)
- Permissão negada em volume/bind
- DNS/serviço não resolve (rede)
- Build lento (cache invalidado)
- Imagem grande (camadas desnecessárias)
- Espaço em disco consumido

Comandos de diagnóstico

```
docker ps -a

docker logs <c>

docker inspect <c>

docker port <c>

docker system df

docker network ls

docker volume ls
```

Regra: observe → formule hipótese → teste.

Limpeza segura

Evite apagar algo que está em uso

Comandos de limpeza

```
# recursos não usados
docker image prune
docker container prune
docker volume prune
docker network prune

# tudo que está “solto”
docker system prune

# (perigoso) inclui imagens não referenciadas
docker system prune -a
```

Boa prática: use primeiro “docker system df” e prune incremental.

Política recomendada

- Em workstations: prune semanal, com orientação
- Em runners CI: limpeza automática por job
- Nunca rode prune agressivo em hosts críticos sem janela

LAB 8 — “Resgate” de ambiente (20–30 min)

Entrega: liberar espaço e restaurar stack

Cenário

- Máquina ficou sem espaço e builds começaram a falhar.
- Objetivo: medir consumo, remover o que não é necessário e garantir que o compose volta a subir.

Roteiro

```
docker system df

docker image prune

docker container prune

docker volume prune

# revalidar
docker system df

# subir stack novamente
docker compose up -d
```

Evite “-a” sem entender impacto.

Checklist

- Você mediu antes e depois
- Você justificou o que removeu
- A stack voltou a subir e responde

Cheat sheet — Comandos essenciais

Para colar no Confluence/Notion do time

CLI rápido

```
# listar
docker ps -a
docker images
docker volume ls
docker network ls

# rodar
docker run --rm -it alpine:3.20 sh
docker logs -f <c>
docker exec -it <c> sh

# build
docker build -t app:1.0 .

# compose
docker compose up -d
docker compose logs -f
docker compose down

# limpeza
docker system df
docker system prune
```

Customize com padrões internos: registry, imagens base e políticas.

Avaliação rápida (10 perguntas)

Use como quiz ao final do treinamento

Perguntas

- 1) Diferença entre imagem e container?
- 2) Para que serve docker inspect?
- 3) Quando usar volume vs bind mount?
- 4) O que é uma bridge network?
- 5) CMD vs ENTRYPOINT — diferença prática?
- 6) Por que multi-stage melhora segurança?
- 7) Como evitar invalidar cache no build?
- 8) Qual o risco do --privileged?
- 9) Qual comando mede uso de disco do Docker?
- 10) O que faz docker compose down -v?

Encerramento

Próximos passos sugeridos

Próximos passos

- Padronizar templates de Dockerfile/Compose por linguagem
- Adicionar scanning e SBOM no CI/CD
- Definir imagens base aprovadas e rotina de atualização
- Documentar runbooks de troubleshooting (incidentes)
- Evoluir para orquestração (Kubernetes) quando necessário

Materiais internos (placeholders)

- Link: Política de containers (segurança)
- Link: Registry corporativo e acesso
- Link: Template de pipelines CI
- Link: Exemplos de stacks (dev/test)