Guia Completo Docker para Desenvolvedores

1. Conceitos Fundamentais

O que é Docker

Docker é uma plataforma de containerização que permite empacotar aplicações e suas dependências em containers leves e portáveis, garantindo que funcionem consistentemente em qualquer ambiente.

Conceitos Básicos

- **Container**: Instância executável de uma imagem, isolada do sistema host
- **Imagem**: Template read-only usado para criar containers
- **Dockerfile**: Arquivo de instruções para construir imagens
- **Registry**: Repositório de imagens (Docker Hub, ECR, etc.)
- **Volume**: Mecanismo para persistir dados gerados pelos containers

Vantagens do Docker

- Consistência entre ambientes (dev, test, prod)
- Isolamento de aplicações
- Portabilidade
- Escalabilidade
- Eficiência de recursos

2. Instalação e Configuração

Instalação

- **Windows**: Docker Desktop
- **macOS**: Docker Desktop
- **Linux**: Docker Engine via package manager

Verificação da Instalação

```bash
docker --version
docker run hello-world
```

Configurações Importantes

- Configurar recursos (CPU, memória)

- Configurar proxy se necessário
- Configurar registry mirrors

3. Comandos Essenciais

Comandos de Imagem

```bash
docker images                # Listar imagens
docker pull <imagem>         # Baixar imagem
docker build -t <nome> .     # Construir imagem
docker rmi <imagem>          # Remover imagem
docker tag <origem> <destino>  # Criar tag
```

Comandos de Container

```bash
docker ps                    # Containers rodando
docker ps -a                 # Todos os containers
docker run <imagem>          # Executar container
docker start <container>     # Iniciar container
docker stop <container>      # Parar container
docker rm <container>        # Remover container
docker exec -it <container> bash # Acessar container
```

Comandos de Logs e Monitoramento

```bash
docker logs <container>      # Ver logs
docker stats                 # Estatísticas em tempo real
docker inspect <container>   # Informações detalhadas
```

4. Dockerfile - Criando Imagens

Estrutura Básica

```dockerfile
```

```
FROM <imagem_base>
WORKDIR /app
COPY . .
RUN <comandos_instalacao>
EXPOSE <porta>
CMD ["comando", "para", "executar"]
```

Instruções Principais

- **FROM**: Imagem base
- **WORKDIR**: Diretório de trabalho
- **COPY/ADD**: Copiar arquivos
- **RUN**: Executar comandos durante build
- **EXPOSE**: Documentar portas expostas
- **ENV**: Variáveis de ambiente
- **CMD/ENTRYPOINT**: Comando padrão de execução

Exemplo Prático (Java/Spring Boot)

dockerfile

```
FROM openjdk:17-jdk-alpine
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

Boas Práticas

- Use imagens base oficiais e mínimas
- Minimize camadas
- Use .dockerignore
- Não execute como root
- Use multi-stage builds para reduzir tamanho

5. Docker Compose - Orquestração Local

O que é Docker Compose

Ferramenta para definir e executar aplicações multi-container usando arquivo YAML.

Arquivo docker-compose.yml

yaml

```yaml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "8080:8080"
    depends_on:
      - database
    environment:
      - DB_HOST=database

  database:
    image: postgres:13
    environment:
      - POSTGRES_DB=myapp
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Comandos Docker Compose

bash

```bash
docker-compose up          # Subir serviços
docker-compose up -d       # Subir em background
docker-compose down        # Parar e remover
docker-compose logs        # Ver logs
docker-compose ps          # Listar serviços
docker-compose exec <serviço> bash # Acessar serviço
```

6. Networking e Volumes

Networking

- **Bridge**: Rede padrão para containers
- **Host**: Usa rede do host
- **None**: Sem conectividade de rede
- **Custom networks**: Redes personalizadas

```bash
docker network create mynetwork
docker run --network=mynetwork <imagem>
```

Volumes

- **Anonymous volumes**: Gerenciados pelo Docker
- **Named volumes**: Volumes nomeados
- **Bind mounts**: Monta diretório do host

```bash
docker volume create myvolume
docker run -v myvolume:/data <imagem>
docker run -v /host/path:/container/path <imagem>
```

7. Ambiente de Desenvolvimento

Containerizando Aplicação Java/Spring Boot

Dockerfile para Desenvolvimento

```dockerfile
FROM openjdk:17-jdk
WORKDIR /app
COPY . .
RUN ./mvnw clean package -DskipTests
EXPOSE 8080
CMD ["java", "-jar", "target/app.jar"]
```

Docker Compose para Stack Completa

```yaml
```

```yaml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "8080:8080"
    environment:
      - SPRING_PROFILES_ACTIVE=docker
      - DATABASE_URL=jdbc:postgresql://db:5432/myapp
    depends_on:
      - db
      - redis
    volumes:
      - .:/app
      - maven_cache:/root/.m2

  db:
    image: postgres:13
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
  maven_cache:
```

Hot Reload e Live Development

- Use bind mounts para código fonte
- Configure ferramentas como Spring DevTools
- Use volumes para cache de dependências

8. Debugging e Troubleshooting

Debugging Containers

```bash
# Acessar container em execução
docker exec -it <container> /bin/bash

# Executar container em modo interativo
docker run -it <imagem> /bin/bash

# Debug de build
docker build --progress=plain .

# Verificar processo dentro do container
docker exec <container> ps aux
```

Problemas Comuns

- **Container sai imediatamente**: Verifique CMD/ENTRYPOINT
- **Erro de conexão**: Verifique networking e portas
- **Problemas de permissão**: Verifique usuário e volumes
- **Container lento**: Verifique recursos alocados

Limpeza e Manutenção

```bash
docker system prune      # Limpar recursos não utilizados
docker image prune       # Remover imagens órfãs
docker container prune   # Remover containers parados
docker volume prune      # Remover volumes não utilizados
```

9. Boas Práticas para Produção

Segurança

- Use imagens oficiais e atualizadas
- Escaneie vulnerabilidades
- Não exponha credenciais
- Execute com usuário não-root
- Use secrets para informações sensíveis

Performance

- Use multi-stage builds
- Minimize layers

- Use .dockerignore adequadamente
- Configure health checks
- Monitore recursos

Exemplo Multi-stage Build

dockerfile

```dockerfile
# Build stage
FROM maven:3.8-openjdk-17 AS build
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests

# Runtime stage
FROM openjdk:17-jre-alpine
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
RUN addgroup -g 1001 -S appuser && \
    adduser -S appuser -u 1001
USER appuser
EXPOSE 8080
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost:8080/actuator/health || exit 1
CMD ["java", "-jar", "app.jar"]
```

10. Integração com CI/CD

Build Automatizado

yaml

```yaml
# GitHub Actions exemplo
name: Build and Push Docker Image
on:
  push:
    branches: [main]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Build and push
        uses: docker/build-push-action@v2
        with:
          push: true
          tags: myregistry/myapp:latest
```

Registry

- Docker Hub (público/privado)
- Amazon ECR
- Google Container Registry
- Azure Container Registry

11. Ferramentas e Extensões Úteis

IDEs e Extensions

- **VS Code**: Docker extension
- **IntelliJ**: Docker plugin
- **Eclipse**: Docker Tooling

Ferramentas CLI

- **docker-compose**: Orquestração local
- **dive**: Analisar camadas de imagem
- **hadolint**: Linter para Dockerfile
- **container-diff**: Comparar imagens

Monitoramento

- **Portainer**: Interface web para gerenciamento
- **cAdvisor**: Monitoramento de containers
- **Grafana + Prometheus**: Métricas avançadas

12. Próximos Passos

Tecnologias Relacionadas

- **Kubernetes**: Orquestração em produção
- **Docker Swarm**: Clustering Docker nativo
- **Helm**: Gerenciamento de pacotes Kubernetes
- **Istio**: Service mesh

Conceitos Avançados

- Microserviços com Docker
- Container orchestration
- Service discovery
- Load balancing
- Distributed tracing

Certificações

- Docker Certified Associate (DCA)
- Kubernetes certifications (CKA, CKAD)
- Cloud provider container certifications