Roteiro Docker para Deploy ASP.NET Core Web API

Do Docker básico ao deploy containerizado

Índice

---

1. Introdução ao Docker

O que é Docker?

Docker é uma plataforma de containerização que permite:

- Empacotar aplicações com todas suas dependências
- Executar de forma consistente em qualquer ambiente
- Isolar aplicações umas das outras
- Escalar horizontalmente com facilidade

Por que usar Docker com .NET?

- Consistência: Mesmo ambiente em dev, test e produção
- Portabilidade: Roda em Windows, Linux, macOS
- Isolamento: Dependências isoladas por container
- Escalabilidade: Fácil escalonamento horizontal
- CI/CD: Integração perfeita com pipelines

Container vs Virtual Machine

Aspecto	Container	Virtual Machine
Startup	Segundos	Minutos

Aspecto	Container	Virtual Machine
Recursos	Compartilha kernel	SO completo
Tamanho	MBs	GBs
Isolamento	Processo	Hardware virtual

---

2. Instalação do Docker

Windows

Docker Desktop (Recomendado)

1. Download: https://www.docker.com/products/docker-desktop
2. Execute o installer
3. Reinicie o computador
4. Configure WSL 2 (se solicitado)

Requisitos Windows

- Windows 10/11 Pro, Enterprise ou Education
- Hyper-V habilitado ou WSL 2
- Virtualização habilitada na BIOS

```
# Verificar se Hyper-V está habilitado
Get-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V

# Habilitar WSL 2 (alternativa)
wsl --install
```

Linux (Ubuntu/Debian)

```
# Remover versões antigas
sudo apt-get remove docker docker-engine docker.io containerd runc

# Instalar dependências
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release

# Adicionar chave GPG oficial do Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

# Adicionar repositório
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
# Instalar Docker Engine
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Adicionar usuário ao grupo docker
sudo usermod -aG docker $USER
newgrp docker
```

macOS

1. Download: Docker Desktop para Mac
2. Arrastar para pasta Applications
3. Executar Docker Desktop

Verificar Instalação

```
# Verificar versão
docker --version
docker-compose --version

# Teste básico
docker run hello-world

# Verificar status
docker ps
docker images
```

---

3. Conceitos Fundamentais

Componentes Principais

Imagem (Image)

- Template read-only para criar containers
- Contém SO, runtime, bibliotecas e aplicação
- Versionada com tags

Container

- Instância executável de uma imagem
- Isolado e efêmero
- Pode ser criado, iniciado, parado, movido e deletado

Dockerfile

- Script com instruções para construir imagem

- Define camadas da imagem
- Cada instrução cria uma nova camada

Registry

- Repositório de imagens Docker
- Docker Hub (público)
- Azure Container Registry (privado)

Comandos Básicos

```
# Listar imagens locais
docker images

# Baixar imagem
docker pull microsoft/dotnet:8.0-aspnet

# Executar container
docker run -p 8080:80 minha-api

# Listar containers rodando
docker ps

# Listar todos containers
docker ps -a

# Parar container
docker stop <container-id>

# Remover container
docker rm <container-id>

# Remover imagem
docker rmi <image-id>

# Ver logs do container
docker logs <container-id>

# Executar comando no container
docker exec -it <container-id> /bin/bash
```

---

4. Preparando o Projeto Web API

Estrutura do Projeto Base

Partindo do projeto criado no guia de instalação:

```
MinhaAPITeste/
├── Controllers/
│   └── WeatherForecastController.cs
├── Properties/
│   └── launchSettings.json
├── appsettings.json
├── appsettings.Production.json
├── Program.cs
├── WeatherForecast.cs
└── MinhaAPITeste.csproj
```

Configurações para Docker

1. Configurar Program.cs para Container

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
  app.UseSwagger();
  app.UseSwaggerUI();
}

// Configurar para aceitar HTTP em containers
if (!app.Environment.IsDevelopment())
{
  app.UseHttpsRedirection();
}

app.UseAuthorization();
app.MapControllers();

// Importante: configurar para escutar em todas as interfaces
app.Run("http://0.0.0.0:80");
```

2. Configurar appsettings.Production.json

```
{
  "Logging": {
  "LogLevel": {
```

```
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
  },
  "AllowedHosts": "*",
  "Urls": "http://0.0.0.0:80"
}
```

3. Adicionar .dockerignore

Crie arquivo `.dockerignore` na raiz do projeto:

```
# Ignorar arquivos desnecessários
bin/
obj/
.git/
.gitignore
.dockerignore
Dockerfile
README.md
.env
.vs/
.vscode/

# Arquivos temporários
*.tmp
*.temp
*~

# Logs
logs/
*.log

# Cache
.cache/
```

---

5. Criando o Dockerfile

Dockerfile Básico

Crie arquivo `Dockerfile` na raiz do projeto:

```
# Usar imagem base oficial do .NET 8
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80

# Usar SDK para build
```

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

# Copiar arquivo de projeto e restaurar dependências
COPY ["MinhaAPITeste.csproj", "."]
RUN dotnet restore "MinhaAPITeste.csproj"

# Copiar código fonte e fazer build
COPY . .
RUN dotnet build "MinhaAPITeste.csproj" -c Release -o /app/build

# Publicar aplicação
FROM build AS publish
RUN dotnet publish "MinhaAPITeste.csproj" -c Release -o /app/publish
/p:UseAppHost=false

# Imagem final
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MinhaAPITeste.dll"]
```

Dockerfile Otimizado (Multi-stage)

```
# ================================================
# Stage 1: Build
# ================================================
FROM mcr.microsoft.com/dotnet/sdk:8.0-alpine AS build
WORKDIR /src

# Copiar apenas os arquivos de projeto primeiro (para cache das dependências)
COPY ["MinhaAPITeste.csproj", "./"]
RUN dotnet restore "MinhaAPITeste.csproj" --runtime linux-musl-x64

# Copiar todo o código fonte
COPY . .

# Build da aplicação
RUN dotnet build "MinhaAPITeste.csproj" \
  --configuration Release \
  --runtime linux-musl-x64 \
  --self-contained false \
  --no-restore \
  --output /app/build

# ================================================
# Stage 2: Publish
# ================================================
FROM build AS publish
RUN dotnet publish "MinhaAPITeste.csproj" \
  --configuration Release \
```

```
    --runtime linux-musl-x64 \
    --self-contained false \
    --no-build \
    --output /app/publish \
    /p:PublishTrimmed=false

# ================================================
# Stage 3: Final Runtime Image
# ================================================
FROM mcr.microsoft.com/dotnet/aspnet:8.0-alpine AS final

# Criar usuário não-root para segurança
RUN addgroup -g 1001 -S appgroup && \
    adduser -S appuser -u 1001 -G appgroup

WORKDIR /app

# Copiar arquivos publicados
COPY --from=publish /app/publish .

# Configurar permissões
RUN chown -R appuser:appgroup /app
USER appuser

# Configurar variáveis de ambiente
ENV ASPNETCORE_ENVIRONMENT=Production
ENV ASPNETCORE_URLS=http://+:8080
ENV DOTNET_RUNNING_IN_CONTAINER=true
ENV DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=true

EXPOSE 8080

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8080/weatherforecast || exit 1

ENTRYPOINT ["dotnet", "MinhaAPITeste.dll"]
```

Comparação das Abordagens

Aspecto	Dockerfile Básico	Dockerfile Otimizado
Tamanho	~200MB	~100MB
Segurança	Root user	Non-root user
Cache	Básico	Otimizado
Health Check	Não	Sim
Produção	OK	Recomendado

6. Construindo a Imagem Docker

Build Básico

```
# Navegar para o diretório do projeto
cd MinhaAPITeste

# Construir imagem
docker build -t minha-api:latest .

# Verificar imagem criada
docker images | grep minha-api
```

Build com Tags e Metadados

```
# Build com tag específica
docker build -t minha-api:v1.0.0 .

# Build com múltiplas tags
docker build -t minha-api:latest -t minha-api:v1.0.0 .

# Build com metadados
docker build \
  --tag minha-api:v1.0.0 \
  --label "version=1.0.0" \
  --label "description=API de exemplo .NET 8" \
  --label "maintainer=seuemail@exemplo.com" \
  .

# Build sem cache (para rebuild completo)
docker build --no-cache -t minha-api:latest .

# Build especificando Dockerfile
docker build -f Dockerfile.production -t minha-api:prod .
```

Analisar a Imagem

```
# Verificar detalhes da imagem
docker inspect minha-api:latest

# Ver histórico de camadas
docker history minha-api:latest

# Verificar tamanho de cada camada
docker images --format "table {{.Repository}}\t{{.Tag}}\t{{.Size}}"
```

```
# Analisar vulnerabilidades (se tiver Docker Scout)
docker scout cves minha-api:latest
```

---

7. Executando Containers

Execução Básica

```
# Executar container
docker run -p 8080:8080 minha-api:latest

# Executar em background (detached)
docker run -d -p 8080:8080 --name minha-api-container minha-api:latest

# Executar com variáveis de ambiente
docker run -d \
  -p 8080:8080 \
  -e ASPNETCORE_ENVIRONMENT=Production \
  -e ASPNETCORE_URLS=http://+:8080 \
  --name minha-api-container \
  minha-api:latest
```

Opções Avançadas

```
# Executar com volume para logs
docker run -d \
  -p 8080:8080 \
  -v $(pwd)/logs:/app/logs \
  --name minha-api-container \
  minha-api:latest

# Executar com restart automático
docker run -d \
  -p 8080:8080 \
  --restart unless-stopped \
  --name minha-api-container \
  minha-api:latest

# Executar com limites de recursos
docker run -d \
  -p 8080:8080 \
  --memory=512m \
  --cpus=1.0 \
  --name minha-api-container \
  minha-api:latest

# Executar com health check customizado
docker run -d \
  -p 8080:8080 \
```

```
  --health-cmd="curl -f http://localhost:8080/weatherforecast || exit 1" \
  --health-interval=30s \
  --health-timeout=3s \
  --health-retries=3 \
  --name minha-api-container \
  minha-api:latest
```

Testar a API

```
# Verificar se container está rodando
docker ps

# Testar endpoint
curl http://localhost:8080/weatherforecast

# Ver logs do container
docker logs minha-api-container

# Logs em tempo real
docker logs -f minha-api-container

# Entrar no container para debug
docker exec -it minha-api-container /bin/sh
```

---

8. Docker Compose

docker-compose.yml Básico

Crie arquivo `docker-compose.yml` na raiz do projeto:

```
version: '3.8'

services:
  api:
  build:
    context: .
    dockerfile: Dockerfile
  image: minha-api:latest
  container_name: minha-api-container
  ports:
    - "8080:8080"
  environment:
    - ASPNETCORE_ENVIRONMENT=Production
    - ASPNETCORE_URLS=http://+:8080
  restart: unless-stopped
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8080/weatherforecast"]
    interval: 30s
```

```
    timeout: 10s
    retries: 3
    start_period: 40s
```

docker-compose.yml Completo

```
version: '3.8'

services:
  api:
  build:
    context: .
    dockerfile: Dockerfile
    args:
    - BUILD_CONFIGURATION=Release
  image: minha-api:${TAG:-latest}
  container_name: ${COMPOSE_PROJECT_NAME:-minha-api}-api
  ports:
    - "${API_PORT:-8080}:8080"
  environment:
    - ASPNETCORE_ENVIRONMENT=${ASPNETCORE_ENVIRONMENT:-Production}
    - ASPNETCORE_URLS=http://+:8080
    - TZ=America/Sao_Paulo
  restart: unless-stopped
  volumes:
    - api-logs:/app/logs
  networks:
    - api-network
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:8080/weatherforecast || exit 1"]
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 40s
  deploy:
    resources:
    limits:
      memory: 512M
      cpus: '1.0'
    reservations:
      memory: 256M
      cpus: '0.5'

  nginx:
  image: nginx:alpine
  container_name: ${COMPOSE_PROJECT_NAME:-minha-api}-nginx
  ports:
    - "${NGINX_PORT:-80}:80"
    - "${NGINX_SSL_PORT:-443}:443"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
```

```
      - ./nginx/ssl:/etc/nginx/ssl:ro
    depends_on:
      api:
      condition: service_healthy
    networks:
      - api-network
    restart: unless-stopped

volumes:
  api-logs:
  driver: local

networks:
  api-network:
  driver: bridge
```

Arquivo .env

Crie arquivo `.env` para variáveis de ambiente:

```
COMPOSE_PROJECT_NAME=minha-api
TAG=v1.0.0

API_PORT=8080
ASPNETCORE_ENVIRONMENT=Production

NGINX_PORT=80
NGINX_SSL_PORT=443

ENABLE_MONITORING=true
```

Comandos Docker Compose

```
docker-compose up -d
docker-compose up -d --build
docker-compose logs -f
docker-compose logs -f api
docker-compose stop
docker-compose down
docker-compose down -v
docker-compose up -d --scale api=3
docker-compose ps
docker-compose exec api /bin/sh
```

---

9. Deploy Local com Docker

Ambiente de Desenvolvimento

docker-compose.dev.yml

```
version: '3.8'

services:
  api:
  build:
    context: .
    dockerfile: Dockerfile
    target: build
  image: minha-api:dev
  container_name: minha-api-dev
  ports:
    - "5000:80"
    - "5001:443"
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_URLS=https://+:443;http://+:80
    - ASPNETCORE_Kestrel__Certificates__Default__Password=password
    - ASPNETCORE_Kestrel__Certificates__Default__Path=/https/aspnetapp.pfx
  volumes:
    - ./src:/src:ro
    - ~/.aspnet/https:/https:ro
    - api-dev-logs:/app/logs
  networks:
    - api-dev-network
  command: ["dotnet", "watch", "run", "--project", "/src"]

volumes:
  api-dev-logs:

networks:
  api-dev-network:
```

Executar ambiente de desenvolvimento

```
dotnet dev-certs https -ep ~/.aspnet/https/aspnetapp.pfx -p password
dotnet dev-certs https --trust

docker-compose -f docker-compose.dev.yml up -d
docker-compose -f docker-compose.dev.yml logs -f api
```

Ambiente de Produção Local

docker-compose.prod.yml

```
```

```yaml
version: '3.8'

services:
  api:
  image: minha-api:latest
  container_name: minha-api-prod
  ports:
    - "8080:8080"
  environment:
    - ASPNETCORE_ENVIRONMENT=Production
    - ASPNETCORE_URLS=http://+:8080
  restart: unless-stopped
  volumes:
    - api-prod-logs:/app/logs
    - api-data:/app/data
  networks:
    - api-prod-network
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:8080/weatherforecast || exit 1"]
    interval: 30s
    timeout: 10s
    retries: 3

  nginx:
  image: nginx:alpine
  container_name: nginx-prod
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/prod.conf:/etc/nginx/conf.d/default.conf:ro
    - ./ssl:/etc/nginx/ssl:ro
  depends_on:
    - api
  networks:
    - api-prod-network
  restart: unless-stopped

volumes:
  api-prod-logs:
  api-data:

networks:
  api-prod-network:
  driver: bridge
```

Configuração Nginx (nginx/prod.conf)

```
upstream api_backend {
  server api:8080;
```

```
}

server {
  listen 80;
  server_name localhost;

  location / {
    proxy_pass http://api_backend;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_connect_timeout 30s;
    proxy_send_timeout 30s;
    proxy_read_timeout 30s;
  }

  location /health {
    proxy_pass http://api_backend/weatherforecast;
    access_log off;
  }
}
```

---

10. Registry e Distribuição

Docker Hub

Fazer Login

```
docker login
echo $DOCKER_HUB_TOKEN | docker login --username $DOCKER_HUB_USERNAME --password-
stdin
```

Push para Docker Hub

```
docker tag minha-api:latest seususuario/minha-api:latest
docker tag minha-api:latest seususuario/minha-api:v1.0.0

docker push seususuario/minha-api:latest
docker push seususuario/minha-api:v1.0.0

docker push seususuario/minha-api --all-tags
```

Azure Container Registry (ACR)

Criar ACR

```
RESOURCE_GROUP="rg-minha-api"
ACR_NAME="minhaapiacr"
LOCATION="East US"

az group create --name $RESOURCE_GROUP --location "$LOCATION"

az acr create \
  --resource-group $RESOURCE_GROUP \
  --name $ACR_NAME \
  --sku Basic \
  --admin-enabled true
```

Push para ACR

```
az acr login --name $ACR_NAME

ACR_SERVER=$(az acr show --name $ACR_NAME --query loginServer --output tsv)

docker tag minha-api:latest $ACR_SERVER/minha-api:latest
docker tag minha-api:latest $ACR_SERVER/minha-api:v1.0.0

docker push $ACR_SERVER/minha-api:latest
docker push $ACR_SERVER/minha-api:v1.0.0

az acr repository list --name $ACR_NAME --output table
```

Registry Privado Local

```
docker run -d \
  -p 5000:5000 \
  --name local-registry \
  --restart unless-stopped \
  -v registry-data:/var/lib/registry \
  registry:2

docker tag minha-api:latest localhost:5000/minha-api:latest
docker push localhost:5000/minha-api:latest

docker pull localhost:5000/minha-api:latest
```

---

11. Deploy na Nuvem

Azure Container Instances (ACI)

```
```

```
RESOURCE_GROUP="rg-minha-api"
CONTAINER_NAME="minha-api-aci"
IMAGE_NAME="$ACR_SERVER/minha-api:latest"

az container create \
  --resource-group $RESOURCE_GROUP \
  --name $CONTAINER_NAME \
  --image $IMAGE_NAME \
  --registry-login-server $ACR_SERVER \
  --registry-username $(az acr credential show --name $ACR_NAME --query username -
o tsv) \
  --registry-password $(az acr credential show --name $ACR_NAME --query
passwords[0].value -o tsv) \
  --dns-name-label $CONTAINER_NAME \
  --ports 8080 \
  --environment-variables ASPNETCORE_ENVIRONMENT=Production \
  --cpu 1 \
  --memory 1.5 \
  --restart-policy Always

az container show \
  --resource-group $RESOURCE_GROUP \
  --name $CONTAINER_NAME \
  --query ipAddress.fqdn \
  --output tsv
```

Azure Container Apps

```
az containerapp env create \
  --name minha-api-env \
  --resource-group $RESOURCE_GROUP \
  --location "$LOCATION"

az containerapp create \
  --name minha-api-app \
  --resource-group $RESOURCE_GROUP \
  --environment minha-api-env \
  --image $ACR_SERVER/minha-api:latest \
  --registry-server $ACR_SERVER \
  --registry-username $(az acr credential show --name $ACR_NAME --query username -
o tsv) \
  --registry-password $(az acr credential show --name $ACR_NAME --query
passwords[0].value -o tsv) \
  --target-port 8080 \
  --ingress external \
  --env-vars ASPNETCORE_ENVIRONMENT=Production \
  --cpu 1.0 \
  --memory 2.0Gi \
  --min-replicas 1 \
  --max-replicas 3
```

```
az containerapp show \
  --name minha-api-app \
  --resource-group $RESOURCE_GROUP \
  --query properties.configuration.ingress.fqdn \
  --output tsv
```

Azure App Service (Linux Container)

```
az appservice plan create \
  --name asp-minha-api \
  --resource-group $RESOURCE_GROUP \
  --is-linux \
  --sku B1

az webapp create \
  --resource-group $RESOURCE_GROUP \
  --plan asp-minha-api \
  --name minha-api-webapp \
  --deployment-container-image-name $ACR_SERVER/minha-api:latest

az webapp config container set \
  --name minha-api-webapp \
  --resource-group $RESOURCE_GROUP \
  --container-image-name $ACR_SERVER/minha-api:latest \
  --container-registry-url https://$ACR_SERVER \
  --container-registry-user $(az acr credential show --name $ACR_NAME --query
username -o tsv) \
  --container-registry-password $(az acr credential show --name $ACR_NAME --query
passwords[0].value -o tsv)

az webapp config appsettings set \
  --resource-group $RESOURCE_GROUP \
  --name minha-api-webapp \
  --settings ASPNETCORE_ENVIRONMENT=Production WEBSITES_PORT=8080
```

---

12. Monitoramento e Logs

Logs do Container

```
docker logs -f minha-api-container
docker logs --tail 50 minha-api-container
docker logs -t minha-api-container
docker logs minha-api-container > api-logs.txt 2>&1
```

Monitoramento de Recursos

```
docker stats minha-api-container
docker stats
docker inspect minha-api-container
docker inspect --format='{{.State.Health.Status}}' minha-api-container
```

Docker Compose Monitoring

```
version: '3.8'

services:
  api:
  # ... configuração da API

  prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
  networks:
    - api-network

  grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin
  volumes:
    - grafana-data:/var/lib/grafana
  networks:
    - api-network

volumes:
  grafana-data:
```

Configuração Prometheus (prometheus/prometheus.yml)

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'api'
  static_configs:
    - targets: ['api:8080']
```

```
    metrics_path: '/metrics'
    scrape_interval: 5s
```

---

13. Boas Práticas e Segurança

Segurança

1. Usuário Não-Root

```
RUN addgroup -g 1001 -S appgroup && \
    adduser -S appuser -u 1001 -G appgroup

USER appuser
```

2. Multi-stage Build

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
# ... build stage

FROM mcr.microsoft.com/dotnet/aspnet:8.0-alpine AS final
# ... final stage com apenas runtime
```

3. Secrets Management

```
echo "minha-senha-secreta" | docker secret create db-password -

secrets:
  db-password:
    external: true

services:
  api:
    secrets:
      - db-password
```

Performance

1. Otimização de Imagem

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0-alpine

RUN apk del .build-dependencies
```

```
COPY --from=publish /app/publish .
```

2. Cache de Camadas

```
COPY ["*.csproj", "./"]
RUN dotnet restore

COPY . .
RUN dotnet build
```

3. Health Checks

```
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:8080/health || exit 1
```

Hardening

1. Scan de Vulnerabilidades

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
  aquasec/trivy:latest image minha-api:latest

docker scout cves minha-api:latest
```

2. Assinatura de Imagens

```
export DOCKER_CONTENT_TRUST=1

docker push seususuario/minha-api:latest
```

3. Limitar Recursos

```
services:
  api:
  deploy:
    resources:
    limits:
      memory: 512M
      cpus: '1.0'
```

```
      reservations:
        memory: 256M
```

---

14. Solução de Problemas

Problemas Comuns

Container não inicia

```
docker logs minha-api-container
docker inspect --format='{{.State.Health}}' minha-api-container
docker run -it --rm minha-api:latest /bin/sh
```

Erro de porta já em uso

```
netstat -tulpn | grep :8080
docker stop $(docker ps -q --filter "publish=8080")
docker run -p 8081:8080 minha-api:latest
```

Problemas de conectividade

```
docker network ls
docker network inspect bridge
docker exec minha-api-container curl http://localhost:8080/weatherforecast
docker exec minha-api-container env
```

Comandos de Debug

```
docker exec -it minha-api-container /bin/sh
docker exec minha-api-container ps aux
docker exec minha-api-container ls -la /app
docker cp minha-api-container:/app/appsettings.json ./debug-appsettings.json
docker inspect minha-api-container | jq '.[0].Config'
```

Limpeza do Sistema

```
docker container prune
docker image prune
docker system prune -a
```

```
docker system df
docker volume prune
```

Checklist de Deploy

Antes do Deploy

- [ ] Dockerfile otimizado com multi-stage build
- [ ] Health check configurado
- [ ] Usuário não-root configurado
- [ ] Variáveis de ambiente configuradas
- [ ] Logs direcionados para stdout/stderr
- [ ] Scan de segurança executado
- [ ] Build funcionando localmente
- [ ] Testes passando em container

Deploy em Produção

- [ ] Registry configurado (ACR/Docker Hub)
- [ ] Secrets gerenciados adequadamente
- [ ] Resources limits definidos
- [ ] Monitoring configurado
- [ ] Backup strategy definida
- [ ] CI/CD pipeline configurado
- [ ] Load balancing (se necessário)
- [ ] SSL/TLS configurado

Conclusão

Parabéns! Você agora domina Docker para deploy de APIs .NET Core!

URLs de Teste

Após seguir este guia, sua API estará disponível em:

- Local: `http://localhost:8080/weatherforecast`
- ACI: `http://minha-api-aci.eastus.azurecontainer.io:8080/weatherforecast`
- Container Apps: `https://minha-api-app.xxx.eastus.azurecontainerapps.io/weatherforecast`

Próximos Passos

1. Implementar CI/CD com GitHub Actions
2. Adicionar banco de dados com Docker Compose
3. Configurar Kubernetes para orquestração
4. Implementar monitoramento completo
5. Configurar auto-scaling

Recursos Adicionais

- Docker Docs: `https://docs.docker.com/`
- Azure Container Instances: `https://docs.microsoft.com/azure/container-instances/`
- Azure Container Apps: `https://docs.microsoft.com/azure/container-apps/`
- .NET in Docker: `https://docs.microsoft.com/dotnet/core/docker/`

Docker + .NET = Combinação perfeita para deploys modernos!

Documento criado em setembro de 2025 - Versão 1.0 Para suporte, consulte a documentação oficial do Docker e Azure