Guia Prático com Docker

1. O que é o Dockerfile?

Um Dockerfile é um arquivo de configuração usado pelo Docker para construir imagens. Ele especifica todas as instruções necessárias para criar uma imagem Docker, incluindo:

- Base do sistema operacional.
- Ferramentas e dependências instaladas.
- Configurações do aplicativo e comandos de execução.

2. Componentes Básicos de um Dockerfile

2.1. Base da Imagem (FROM)

A instrução FROM define a imagem base que será usada para construir a nova imagem.

Exemplo:

FROM openjdk:17-jdk-slim

- **openjdk**: Nome da imagem base.
- **17-jdk-slim**: Tag da imagem, indicando a versão do JDK (Java Development Kit) e que esta é uma versão reduzida.

Dica:

- Imagens **slim** são menores e otimizadas para produção.
- Imagens completas são úteis durante o desenvolvimento, quando mais ferramentas podem ser necessárias.

2.2. Metadados (LABEL)

A instrução LABEL permite adicionar informações sobre a imagem, como o mantenedor.

Exemplo:

LABEL maintainer="Samuel Rógenes"

2.3. Copiar Arquivos para a Imagem (COPY)

O comando COPY copia arquivos ou diretórios do sistema local para o sistema de arquivos da imagem Docker.

Exemplo:

COPY target/accounts-0.0.1-SNAPSHOT.jar accounts-0.0.1-SNAPSHOT.jar

- **target/accounts-0.0.1-SNAPSHOT.jar**: Localização do arquivo no host.
- **accounts-0.0.1-SNAPSHOT.jar**: Nome e local onde o arquivo será salvo dentro da imagem.

2.4. Configurar o Comando de Entrada (ENTRYPOINT)

A instrução ENTRYPOINT define o comando que será executado quando um container for iniciado a partir da imagem.

Exemplo:

ENTRYPOINT ["java", "-jar", "accounts-0.0.1-SNAPSHOT.jar"]

- **java**: Executa o comando Java.
- **-jar**: Indica que será executado um arquivo JAR.
- **accounts-0.0.1-SNAPSHOT.jar**: Nome do arquivo a ser executado.

3. Criando a Imagem e o Container

3.1. Gerar o Arquivo JAR do Projeto

Certifique-se de que o projeto foi empacotado corretamente antes de criar a imagem.

Use o Maven para gerar o arquivo JAR:

mvn clean install

Este comando criará o JAR dentro da pasta target.

3.2. Criar a Imagem

Use o comando docker build para construir a imagem Docker:

docker build -t accounts-service .

- **-t accounts-service**: Nome da imagem gerada.
- **.**: Indica que o Dockerfile está no diretório atual.

3.3. Executar o Container

Depois de criar a imagem, você pode iniciar um container com o comando docker run:

docker run -p 8080:8080 accounts-service

- **-p 8080:8080**: Mapeia a porta 8080 do host para a porta 8080 do container.
- **accounts-service**: Nome da imagem a ser usada.

4. Fluxo Completo do Docker

1. Criar o arquivo JAR do projeto:

mvn clean install

2. Criar a imagem:

docker build -t accounts-service .

3. Executar o container:

docker run -p 8080:8080 accounts-service

5. Dicas e Boas Práticas

1. **Organização de Imagens e Tags:**

Inclua versões ao nome da imagem para facilitar o gerenciamento:

docker build -t accounts-service:v1.0 .

2. **Multi-stage Builds:**

Para reduzir o tamanho da imagem, use *multi-stage builds*.

Exemplo:

```
FROM maven:3.8.7-openjdk-17 AS build
WORKDIR /app
COPY . .
RUN mvn clean install

FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/accounts-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

3. **Verificar o Status dos Containers:**

a. Listar containers em execução:

docker ps

b. Parar um container:

docker stop <container_id>

4. **Manter Imagens Atualizadas:**

Antes de usar uma imagem base, verifique se há uma versão mais recente:

docker pull openjdk:17-jdk-slim

5. Gerando e Executando Imagens com Docker

5.1. Criando uma Imagem Docker

O comando abaixo é usado para gerar uma imagem a partir de um Dockerfile:

docker build . -t nome-da-imagem:tag

- **-t nome-da-imagem:tag**: Define o nome e a tag da imagem.
- **.**: Indica que o Dockerfile está no diretório atual.

Exemplo:

docker build . -t samuelrcf/accounts:1.0

5.2. Executando um Contêiner

Depois de criar uma imagem, você pode iniciar um contêiner com os comandos abaixo:

Comando básico:

docker run -p 8080:8080 samuelrcf/accounts:1.0

- **-p 8080:8080**: Mapeia a porta do host (primeiro valor) para a porta do contêiner (segundo valor).
- **samuelrcf/accounts:1.0**: Nome e tag da imagem.

Executando no modo desanexado:

docker run -d -p 8080:8080 samuelrcf/accounts:1.0

- **-d**: Executa o contêiner em segundo plano, permitindo que o terminal permaneça livre.

Notas importantes:

- Múltiplos contêineres do mesmo microsserviço devem compartilhar a mesma **porta interna** (configuração da aplicação) para facilitar o balanceamento de carga.
- Cada contêiner, dentro da mesma rede Docker, se comporta como uma máquina em uma rede local.

6. Criando Imagens com Buildpacks

Os **Buildpacks** geram imagens Docker automaticamente a partir do código-fonte, aplicando práticas recomendadas de produção.

No contexto do Spring Boot, isso é integrado ao **Spring Boot Maven Plugin**.

Configuração no pom.xml:

```
<build>
   <plugins>
      <plugin>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-maven-plugin</artifactId>
         <configuration>
            <image>
               <name>eazybytes/${project.artifactId}:1.0</name>
            </image>
         </configuration>
      </plugin>
   </plugins>
</build>
```

Gerando a Imagem:

No terminal, execute:

mvn spring-boot:build-image

Este comando cria a imagem com base no código e na configuração.

Executando a Imagem:

Depois de gerada, use o comando docker run:

docker run -p 8080:8080 eazybytes/accounts:1.0

7. Gerando Imagens com o Google Jib

O **Google Jib** é um plugin para Maven que simplifica a criação de imagens Docker diretamente, sem a necessidade de escrever um Dockerfile.

Configuração no pom.xml:

```
<plugin>
   <groupId>com.google.cloud.tools</groupId>
   <artifactId>jib-maven-plugin</artifactId>
   <version>3.4.3</version>
   <configuration>
      <to>
         <image>myimage</image>
      </to>
   </configuration>
</plugin>
```

Gerando a Imagem:

Execute o comando:

mvn compile jib:dockerBuild

Isso criará a imagem no Docker local com o nome especificado no <image>.

Executando a Imagem:

Assim como antes, use o comando docker run:

docker run -p 8080:8080 myimage

8. Comparação entre Buildpacks e Jib

Aspecto	Buildpacks	Jib
Configuração	Automatizado pelo Spring Boot Maven Plugin	Requer configuração explícita no `pom.xml`.
Requisito de Docker	Docker precisa estar instalado.	Não requer Docker instalado localmente.
Praticidade	Ideal para quem já usa o Spring Boot Maven Plugin.	Focado em eficiência e integração com CI/CD.

9. Boas Práticas no Uso de Docker

1. **Usar Imagens Slim para Produção:**
2. As imagens slim são menores e mais seguras, ideais para ambientes de produção.
3. **Gerenciamento de Tags:**

Utilize tags significativas para rastrear versões de imagens.

Exemplo:

docker build -t accounts-service:1.0 .

4. **Multi-Stage Builds:**

Reduza o tamanho da imagem final utilizando *multi-stage builds* no Dockerfile.

5. **Redes Docker:**

Configure redes específicas para comunicação entre contêineres:

docker network create my-network
docker run --network my-network -p 8080:8080 myimage

6. **Limpeza de Imagens e Contêineres Antigos:**

Para liberar espaço, remova imagens e contêineres que não estão em uso:

docker system prune -a

10. Publicando Imagens no Docker Hub

Para enviar imagens Docker ao Docker Hub, use o comando docker push com o caminho completo da imagem:

Exemplo:

docker image push docker.io/samuelrcf22/accounts:1.0
docker image push docker.io/samuelrcf22/message:s13

- **docker.io**: Indica que a imagem será enviada ao Docker Hub.
- **samuelrcf22**: Nome do usuário no Docker Hub.
- **accounts:1.0**: Nome da imagem e tag a ser publicada.

11. Estrutura do docker-compose.yml

O Docker Compose facilita a definição e execução de aplicativos que utilizam múltiplos contêineres. Abaixo, um exemplo com microsserviços:

```
services:
  cards:
    image: "samuelrcf/cards:1.0"
    container_name: cards-ms
    ports:
      - "8082:8082"
    deploy:
      resources:
        limits:
          memory: 700m
    networks:
      - eazybank

networks:
  eazybank:
    driver: "bridge"
```

Seções do Arquivo

1. **services**: Lista e configura os serviços (contêineres).
    a. **cards**: Nome do serviço (contêiner).
    b. **image**: Especifica a imagem a ser usada.
    c. **container_name**: Define o nome do contêiner.
    d. **ports**: Mapeia portas do host para o contêiner.
    e. **deploy**: Define limites de recursos (relevante no Docker Swarm).
        i. **memory**: Limita o uso de memória do contêiner.
    f. **networks**: Conecta o contêiner a redes Docker.
2. **networks**: Define as redes usadas pelos contêineres.
    a. **eazybank**: Nome da rede.

b. **driver**: Define o tipo da rede (ex.: bridge).

12. Comandos do Docker Compose

Iniciar Contêineres

docker compose up -d

- **-d**: Executa em modo desanexado (em segundo plano).

Parar e Remover Contêineres

docker compose down

Parar Temporariamente os Contêineres
docker compose stop

- **Nota:** Não é recomendável usar stop como prática, pois pode deixar o ambiente inconsistente.

Reiniciar Contêineres

docker compose start

13. Criando um Contêiner MySQL

Para criar um contêiner de banco de dados MySQL com Docker:

docker run -p 3307:3306 --name loansdb -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=loansdb -d mysql

Parâmetros:

- **-p 3307:3306**: Mapeia a porta 3306 do contêiner para 3307 no host.
- **--name loansdb**: Nome do contêiner.
- **-e MYSQL_ROOT_PASSWORD=root**: Define a senha do usuário root.
- **-e MYSQL_DATABASE=loansdb**: Cria automaticamente o banco de dados loansdb.
- **-d**: Executa em segundo plano.

14. Configurando Healthchecks no Docker Compose

O **healthcheck** monitora a saúde dos contêineres em execução. Abaixo, um exemplo para um banco de dados MySQL:

```
services:
  accountsdb:
    image: mysql
    container_name: accountsdb
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: accountsdb
    ports:
      - "3306:3306"
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 3
      start_period: 10s
```

Campos do Healthcheck:

1. **test**: O comando para verificar a saúde do contêiner.
    a. No exemplo, verifica a disponibilidade do banco de dados.
2. **interval**: Intervalo entre as execuções do comando.
3. **timeout**: Tempo limite para que o comando retorne sucesso.
4. **retries**: Número de tentativas antes de marcar como unhealthy.
5. **start_period**: Tempo de espera antes de iniciar o healthcheck após o contêiner ser iniciado.

15. Comportamento do Healthcheck

- Após o start_period, o healthcheck executa o comando no test periodicamente.
- Se o comando retornar sucesso dentro do timeout, o contêiner é considerado healthy.
- Caso contrário, após exceder o número de retries, o contêiner é marcado como unhealthy.

16. Uso de Variáveis de Ambiente no Docker Compose

As variáveis de ambiente são amplamente usadas em configurações Docker Compose para ajustar o comportamento das aplicações e serviços.

SPRING_PROFILES_ACTIVE

- Especifica quais perfis de configuração estão ativos.
- Exemplos de perfis:
    - default: Configurações padrão.
    - dev: Configurações específicas para o ambiente de desenvolvimento.
    - prod: Configurações para produção.

SPRING_CONFIG_IMPORT

- Permite importar configurações externas.
- No contexto do **Spring Cloud Config Server**, uma URL no formato http://configserver:8083/ será usada para buscar configurações.
    - configserver: Nome do serviço no docker-compose.yml.
    - 8083: Porta onde o Config Server está exposto.

Exemplo no Docker Compose:

```
services:
  accounts:
    image: "samuelrcf/accounts:1.0"
    environment:
      SPRING_PROFILES_ACTIVE: dev
      SPRING_CONFIG_IMPORT: "configserver:http://configserver:8083/"
      SPRING_DATASOURCE_USERNAME: app_user
      SPRING_DATASOURCE_PASSWORD: app_password
    networks:
      - eazybank
```

17. Configuração de Usuários no Banco de Dados

É recomendado criar usuários específicos com permissões limitadas para cada serviço ou aplicação, evitando o uso do usuário root em produção.

Exemplo de Configuração no Docker Compose:

```
services:
  accountsdb:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: accountsdb
      MYSQL_USER: app_user
      MYSQL_PASSWORD: app_password
```

```
  ports:
    - "3306:3306"
```

- **MYSQL_USER**: Define o usuário adicional para a aplicação.
- **MYSQL_PASSWORD**: Define a senha para este usuário.

18. Executando Serviços com Docker

Redis

Redis é um banco de dados em memória usado para caching e mensagens.

```
docker run -p 6379:6379 --name srredis -d redis
```

- **-p 6379:6379**: Mapeia a porta do Redis.
- **--name srredis**: Nome do contêiner.
- **-d**: Executa em segundo plano.

Keycloak

Keycloak é uma solução para gerenciar autenticação e autorização.

```
docker run -d -p 7080:8080 \
  -e KC_BOOTSTRAP_ADMIN_USERNAME=admin \
  -e KC_BOOTSTRAP_ADMIN_PASSWORD=admin \
  quay.io/keycloak/keycloak:26.0.5 start-dev
```

- **KC_BOOTSTRAP_ADMIN_USERNAME** e **KC_BOOTSTRAP_ADMIN_PASSWORD**: Definem as credenciais do administrador.
- **start-dev**: Executa o Keycloak em modo de desenvolvimento.

RabbitMQ

RabbitMQ é um sistema de mensagens que suporta filas assíncronas.

```
docker run -it --rm --name rabbitmq \
  -p 5672:5672 -p 15672:15672 \
  rabbitmq:4.0-management
```

- **-p 5672:5672**: Porta padrão para comunicação entre aplicações.
- **-p 15672:15672**: Porta para acessar a interface de gerenciamento do RabbitMQ.

- **-it --rm**: Executa o contêiner interativamente e remove-o após a execução.

Nota: Verifique as versões mais recentes do RabbitMQ.

19. Comunicação entre Serviços no Docker Compose

No Docker Compose, os serviços podem se comunicar pelo **nome do serviço** se estiverem na mesma rede.

Exemplo:

No docker-compose.yml:

```
services:
  accounts:
    image: "samuelrcf/accounts:1.0"
    networks:
      - eazybank
  configserver:
    image: "samuelrcf/configserver:1.0"
    networks:
      - eazybank
networks:
  eazybank:
    driver: "bridge"
```

Neste exemplo:

- O serviço accounts pode se conectar ao configserver usando http://configserver:8083/.