DevOps

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

Containers e Docker

O Problema: "Na Minha Máquina Funciona..."

O Cenário Clássico:

- Um desenvolvedor cria o código em seu notebook (Dev).
- Ele funciona perfeitamente.
- Quando o código é movido para o servidor de testes (Ops), ele quebra!

O Problema: "Na Minha Máquina Funciona..."

Causas Comuns:

- Versão diferente do Java (JDK 8 vs. JDK 17).
- Ausência de uma biblioteca específica.
- Configurações diferentes do sistema operacional.
- Variáveis de ambiente ausentes.

Resultado: O time de Operações gasta horas tentando replicar o ambiente do Desenvolvedor.

O Container: O Padrão Universal

Analogia da Carga: Imagine que o software é uma **carga** e o Container é o **contêiner de navio**.

- Não importa se a carga é café, carro ou grãos, ela é empacotada em um contêiner de tamanho padrão.
- O navio, caminhão ou trem (o servidor) sabe exatamente como transportar contêineres, sem se preocupar com o que está dentro.

O Container: O Padrão Universal

Definição Técnica: Um container é um pacote padronizado que inclui tudo que o software precisa para rodar:

- O Código.
- O Runtime (ex: o JDK para Java).
- As Bibliotecas de Sistema.
- As Configurações.

Princípio: O container garante que a aplicação rode exatamente da mesma forma em qualquer ambiente.

Docker: A Ferramenta Mais Popular

Definição: Docker é a plataforma (ou "motor") mais popular para construir, distribuir e executar containers.

Componentes Principais:

- **Docker Engine:** O software que roda os containers nos servidores.
- **Imagens (Images):** O modelo estático e "pronto para ler" do seu container (a "receita" final).
- **Docker Hub:** O repositório central onde as imagens podem ser armazenadas e compartilhadas (como o GitHub, mas para imagens).

Slogan: Construa uma vez, execute em qualquer lugar.

A Grande Diferença: Compartilhamento vs. Isolamento Total

Máquina Virtual (VM):

- Isolamento total: Cada VM tem seu próprio Sistema Operacional (SO) completo e pesado.
- Desvantagem: Lenta para iniciar, consome muita RAM e espaço em disco (Gigabytes).

Container (Docker):

- Compartilhamento: O container compartilha o kernel do SO do host.
- Vantagem: Leve, inicia em segundos, consome menos recursos (Megabytes).
- Resultado: Você pode rodar muito mais containers em um único servidor do que VMs.

A Grande Diferença: Compartilhamento vs. Isolamento Total

Característica	Máquina Virtual (VM)	Container (Docker)
Recursos	Pesado (SO dedicado)	Leve (Compartilha o SO do host)
Isolamento	Forte (Hardware Virtualizado)	Médio (Processos Isolados)
Inicialização	Minutos	Segundos

Dockerfile: A Receita para a Imagem

Conceito: O Dockerfile é um simples arquivo de texto que contém todas as instruções passo a passo necessárias para construir uma Imagem Docker.

Infraestrutura como Código: A descrição do ambiente é versionada junto com o código-fonte (Git).

Exemplo de Linhas (Linguagem):

- FROM openjdk:17-jdk-slim (Começa com uma imagem Java base)
- COPY ./app.jar /usr/app/ (Copia o código compilado)
- EXPOSE 8080 (Abre uma porta)
- CMD ["java", "-jar", "app.jar"] (Define o comando de execução)

Containers: Essenciais para o DevOps

Consistência: Resolve o problema "Funciona na minha máquina", garantindo que a aplicação se comporte de forma idêntica em Desenvolvimento, Teste e Produção.

Imutabilidade: As imagens Docker, uma vez construídas, não mudam. O que passou no teste é exatamente o que vai para a produção.

Portabilidade: Um container Docker pode ser movido entre diferentes provedores de nuvem (AWS, Azure, Google Cloud) ou data centers privados sem reconfiguração.

Escalabilidade (Kubernetes): É o requisito básico para orquestradores como Kubernetes, que gerenciam milhares de containers automaticamente.

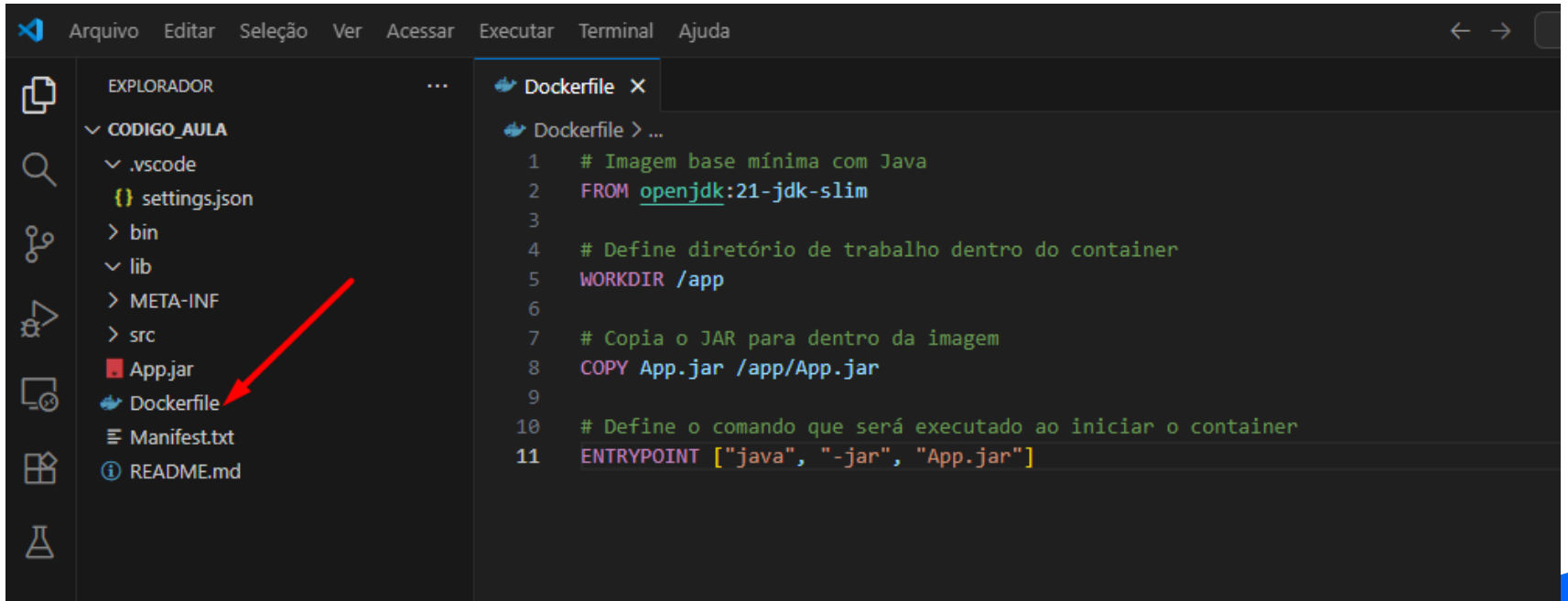
Prática com Docker

Vamos simular o cenário mais comum em produção:

- O desenvolvedor compila o código na máquina dele (ou em uma etapa anterior do CI, que não é o Docker).
- O Docker apenas pega o arquivo JAR final e o executa.

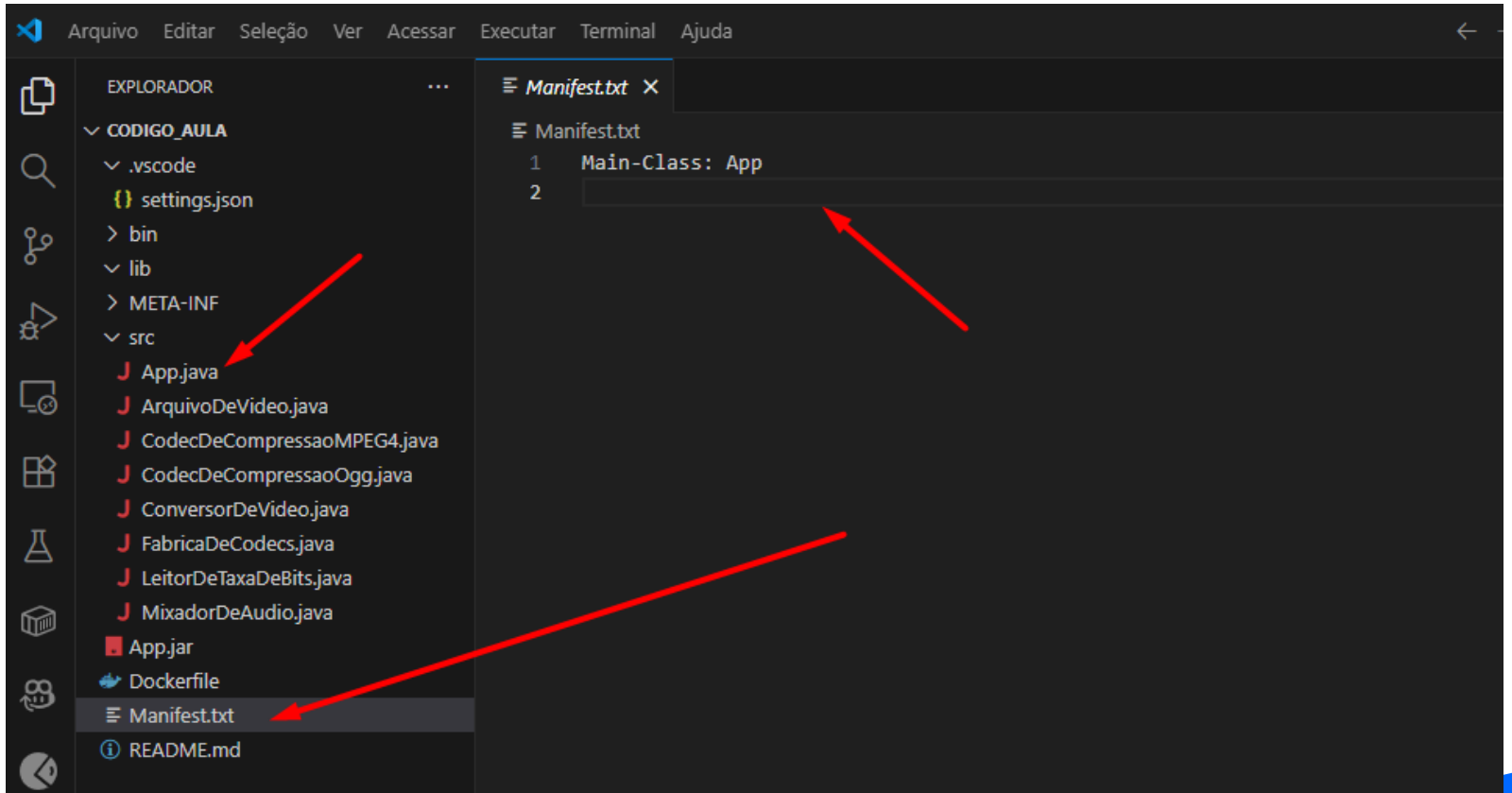
Prática com Docker

- Para essa prática use um programa em java produzido anteriormente. (Por exemplo na matéria de Design Patterns)
- Crie o arquivo Dockerfile:



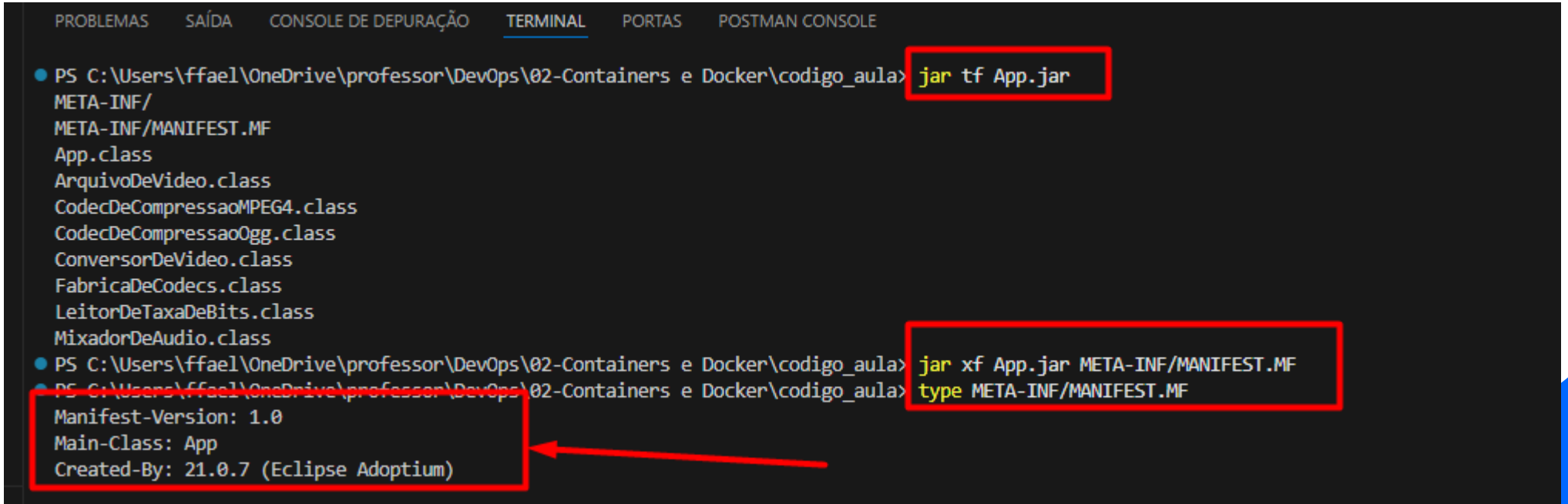
Prática com Docker

- Crie o arquivo Manifest.txt para indicar a sua classe “main”.



Prática com Docker

- Compilar o código:
  - javac -d bin src/*.java
- Crie o JAR com o manifesto
  - jar cfm App.jar Manifest.txt -C bin .
- Confirme se o Manifesto foi embutido corretamente:



Prática com Docker

- Construir e Rodar o Docker:
  - docker build -t app-java-simples .
  - docker run app-java-simples

Exercício em grupo

1. Conforme a prática com Docker realizada em aula, replique os passos em seu computador e tire um print da tela com o último comando executado no Docker. Envie esse print juntamente com um pequeno parágrafo explicando em quais situações essa abordagem com Docker pode ser útil dentro de uma empresa.

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional.8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.