# FIAP - FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

**DEVOPS TOOLS & CLOUD COMPUTING**

* Marcus Martins Leite

# TURMA:

* 2TDSPZ

# ALUNOS:

* Felipe Pinheiro RM 550244

# PROJETO:

* Apostila sobre Dockerfile e Docker Compose

# O DESAFIO:

Realizar a apostila explicando os exercícios e mostrando os resutados

# SUMÁRIO

[INTRODUÇÃO 4](#_bookmark0)

[DESCRIÇÃO DO PROBLEMA A RESOLVER 4](#_bookmark1)

[DESCRIÇÃO DOS OBJETIVOS DA SOLUÇÃO IDEALIZADA 5](#_bookmark2)

[DEFINIÇÃO DO PÚBLICO-ALVO 5](#_bookmark3)

[ESTUDO DE PRODUTOS NO MERCADO 6](#_bookmark4)

[AVALIAÇÃO DO POTENCIAL DE MERCADO 7](#_bookmark5)

[BUSINESS MODEL CANVAS 7](#_bookmark6)

# 1: DOCKERFILE COM INSTALAÇÃO DO PYTHON

Nesta atividade, iremos criar uma imagem que faz a instalação do Python em uma distribuição Ubuntu do Linux baseado em um arquivo chamado Dockerfile.

Passo 1: Abrir o Bloco de notas digitando os comandos abaixo

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Comandos utilizados:**

* **Comando FROM:** Informa ao Docker qual é o sistema que vai rodar a aplicação. Neste caso é o Ubuntu;
* **Comando RUN:** Informa ao Docker para rodar o comando que vem a seguir;
* **Comando -y:** significa para dar “yes” automaticamente na atualização/instalação.

Passo 2: Abrir o terminal (CMD ou PowerShell) no mesmo diretório onde o arquivo Dockerfile está salvo e digitar:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

OBS:

* O comando -t significa tag, serve para dar um nome à imagem Docker.
* O comando “.” ao final do script significa para apontar que o Dockerfile está no mesmo diretório

O resultado esperado é esse:

Texto

Descrição gerada automaticamente

Passo 3: Para verificar se é possível usar o python através dessa imagem, vamos colocá-lo em um container. Basta inserir:

Texto

Descrição gerada automaticamente

E ele funciona! Se dermos o comando “python3”, podemos até escrever livremente em python:

Texto

Descrição gerada automaticamente

# 1.1: RECRIANDO A IMAGEM COM O TOUCH

Pois bem, vamos recriar a imagem de um jeito mais eficiente. Vamos complementar nosso Dockerfile adicionando uma linha usando o comando “touch”:

Texto

Descrição gerada automaticamente

Agora, vamos rodar nossa imagem com o **docker build -t ubuntu\_python .**

**Texto

Descrição gerada automaticamente**

Vamos executar um novo container com o:

* **docker container run -it ubuntu\_python /bin/bash**

**Texto

Descrição gerada automaticamente**

# 2. ADICIONANDO ARQUIVOS NO DOCKERFILE

Passo 1: No dockerfile, adicione o arquivo desejado com o comando “ADD”:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Passo 2: Construir uma imagem usando o “**docker build -t <nome> .**”:

Texto

Descrição gerada automaticamente

Passo 3: Rodar o container usando o “**docker container run --name testeadd -it teste /bin/bash**”:

Tela de um aparelho eletrônico

Descrição gerada automaticamente com confiança baixa

**3. COMANDO EXPOSE**

Passo 1: Crie o arquivo dockerfile com esses comandos:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Passo 2: Construa uma imagem usando o “**docker build -t <nome> .**”:

Texto

Descrição gerada automaticamente

Passo 3: Insepcione a imagem com: “**docker image inspect –format= <nome>**”:

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

Passo 4: Em outro terminal, digite: “**docker container run --name testeexpose -it teste bash**”

Texto

Descrição gerada automaticamente

**4. EXPOSE COM SINALIZADOR DE PUBLICAÇÃO**

Passo 1: Escreva esse comando **“docker run -d –name nginx-server -p 80:80 nginx”**

**Texto

Descrição gerada automaticamente**

Passo 2: Abra o navegador e acesse: [**http://localhost:80**](http://localhost:80)

Texto

Descrição gerada automaticamente

Passo 3: Parar o container usando **“docker container stop nginx-server”** e depois remover o container usando **“docker container rm nginx-server”**

Tela de um aparelho eletrônico

Descrição gerada automaticamente com confiança baixa

**5. COMANDO WORKDIR**

Workdir: Responsável por definir o local padrão de trabalho no Container, onde serão executados os comandos, também define o lugar padrão quando for executado o Container com a execução “-it”

Passo 1: Crie o arquivo no seu Host, no Terminal execute o comando:

**echo { "nome": "Robert Plant", "banda": "Led Zeppelin" } > arquivo-host.json**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Passo 2: Coloque no seu Dockerfile:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Passo 3: Construa a imagem com o **docker build -t teste .**

Texto

Descrição gerada automaticamente

Passo 4: Rode o container com o comando **docker container run --name testeworkdir -it teste bash**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**6. COMANDO CMD**

O comando CMD define um comando padrão que vai ser executado no momento que é executado o Container

Passo 1: Digite isso no seu Dockerfile:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Passo 2: Construa a sua imagem usando **docker build -t testecmd .**

Texto

Descrição gerada automaticamente

Passo 3: Rode o container com **docker container run –rm testecmd**



**OBS:** O comando –rm limpa automaticamente o Container e remove os arquivos quando for encerrado

**7. EXEMPLO HELLO WORLD COM CMD**

1. Clone o projeto com **git clone https://github.com/profjoaomenk/Docker-NodeJs-HelloWorld.git**

Texto

Descrição gerada automaticamente

2. Entre no diretório Docker-NodeJs-HelloWorld usando o **cd** e construa a imagem com o **docker build -t nodehelloworld .**

Texto

Descrição gerada automaticamente

Passo 3: Execute o container com esse comando:

* **docker container run -d -p 3030:3030 --name apphello nodehelloworld**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

É importante notar que podemos parar o container usando o **docker container stop <nomedaimagem>** e também iniciar ele usando o **docker container start <nomedaimagem>**.

**8. COMANDO ENTRYPOINT**

Serve como ponto de entrada para seu Container, basicamente é o que ele vai fazer quando for inciado.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Construindo a imagem com o **docker build -t ptoentrada .**

**Texto

Descrição gerada automaticamente**

Executando o container com **docker container run --rm ptoentrada**

**Texto

Descrição gerada automaticamente**

**9. ENTRYPOINT E CMD JUNTOS**

ENTRYPOINT deve ser definido ao usar o Container como um executável. Já o CMD deve ser usado como uma forma de definir argumentos padrão para um comando ENTRYPOINT ou para executar um comando ad hoc em um Container. Ele será substituído ao executar o Container com argumentos alternativos

Texto

Descrição gerada automaticamente

Construindo a imagem: **docker build -t teste .**

**Texto

Descrição gerada automaticamente**

Executando o container com nome de **appentryecmd**

**Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente**

Removendo o container anterior e testando com um nome alternativo

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média**

Comandos utilizados:

**docker container rm appentryecmd**

**docker container run --name appentryecmdteste "Felipe"**

**10. COMANDO ARG**

A instrução ARG define uma variável que os usuários podem passar em tempo de compilação para o construtor da Imagem (comando docker build) usando o sinalizador: --build-arg <varname>=<value>

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Construindo a imagem com **docker build -t arg-demo .**

Texto

Descrição gerada automaticamente

Rodando o container

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Agora vamos substituir o valor padrão:

Texto

Descrição gerada automaticamente