**Universidade Estadual do Oeste do Paraná**

Campus Cascavel

Centro de Ciências Exatas e Tecnológicas

*Curso de Bacharelado em Ciência da Computação*

**AVALIAÇÃO PRÁTICA**

**Unidade Curricular: Tecnologias Para Desenvolvimento de Sistemas (2025)**

**Professor: Leonardo Medeiros**

**VALOR:** máximo de 100 PONTOS

**NOME:** Pedro Moraes Michalski

**AP02 – Implementação de Sistemas em Containers RunC, LXC, Docker e Podman.**

**Conteúdo: Tecnologias de Desenvolvimento e Operação de Sistemas**

* Sistema Operacional GNU com Linux
* Interface de Texto
* Containers: RunC, LXC, Docker e Podman
* Sistemas de Controle de Versão: Git

**Objetivos**

* + Implementar Sistemas em Containers

◦ RunC

◦ LXC

◦ Docker

◦ Podman

* + Criar e compartilhar repositório do projeto Git

**Descrição de procedimentos**

**Parte I (RunC)**

O runc é um runtime de contêineres — basicamente, um programa que sabe criar e executar contêineres seguindo a especificação OCI (Open Container Initiative).

Para começar com os procedimentos, temos que instalar o runc, verificar a instalação.

**Comandos (Os comandos estão com sudo, por conta da permissão de adiministrador)**

**$ sudo apt install runc**

**$ runc –version**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem I – Instalação do runc**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Image II - Verificação de instalação**

Agora, devemos criar alguns diretórios, a fim de utilizarmos para trabalhar com os containers. O qual criaremos um diretório principal “runc-dire”, e dentro desse diretório criamos “rootfs” que servirá como pasta de root systemfile. Assim utilizando o comando “sudo runc spec”, para gerar o arquivo “config.json” que é baseado na especificação OCI, para enfim verificar sua criação.

**Comandos:**

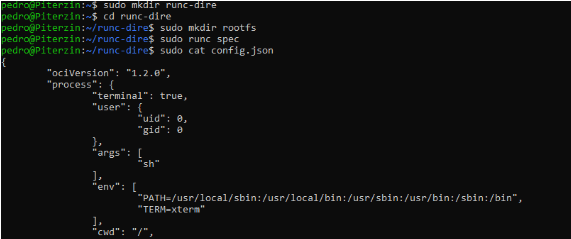
**$ sudo mkdir runc-dire**

**$ cd runc-dire**

**$ sudo mkdir rootfs**

**$ sudo runc spec**

**$ sudo cat config.json**



**Image III – Criação de repositórios e criação do config.json**

O proximo passo a se fazer, é baixar a imagem Alpine via Docker. Cria um container a partir da imagem alpine, sem executar. Exporta o sistema de arquivos desse container em formato .tar, extraimos para dentro de rootfs.

**Comandos:**

**$ sudo sh –c `docker export $(docker create alpine) | tar –C rootfs -xvf`**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Image IV – Imagem alpine via docker**

Nesse momentos rodamos o container e verificamos se está tudo certo, o runc procura no diretório atual um config.json.

**Comandos:**

**$ sudo runc run cgl**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem V – Run do container**

**Parte II (lxc/lxd)**

Após, termos criado um diretório container, movemos o diretório “runc-dire” para dentro do container (Se possível, já comece com o runc-dire dentro desse diretório, esqueci de organizar antes). Dessa forma, começamos a instalação do lxc

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem VI – Instalação do lxc**

Nesse momento, criamos os containers usando lxc (Linux Containers) c1 e c2, porém falhou a c2 utilizando debian, então tive que utilizar uma instalação a parte. Também configurando o container c2.

**Comandos:**

**$ sudo lxc-create –n c1 –t busybox**

**$ sudo lxc-create –n c2 –t debian**

**$ sudo lxc-create –n container-debian –t download**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem VII – Criação dos containers lxc**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem VIII – Configuração do container.**

A seguir, iniciamos os container c1 e c2, e conectamos o terminal a um processo dentro do container, basicamente entrando no container.

**Comandos:**

**$ sudo lxc-start –n c1**

**$ sudo lxc-attach –n c1**

**$ sudo lxc-start –n c2**

**$ sudo lxc-attach –n c2**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem IX – Run e attach do container c1**

Tela de computador com letras e números em fundo preto

O conteúdo gerado por IA pode estar incorreto.**Imagem X – Run e attach do container c2**

Agora, realizamos a instalação do libvirt-daemon-systems, para gerenciar máquinas virtuais e contêineres LXC pelo virt-manager (gráfico) ou virsh (CLI). Posteriormente finalizando os containers.

**Comandos:**

**$ sudo apt install libvirt-daemon-system virt manager virtinst libvirt-clients libvirt-daemon-driver-lxc**

**$ sudo lxc-stop –n c1**

**$ sudo lxc-stop –n c2**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XI – Instalação do libvirt-daemon-systems.**

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XII – Stop dos containers c1 e c2.**

Para continuar com os processos, verifique se o apt está atualizado, realizamos a instalação do snapd, e adicionamos o usuário atual a um grupo lxd, o que permitirá executar os comandos lxd.

**Comandos:**

**$ sudo apt update**

**$ sudo apt install snapd –y**

**$ sudo usermod –aG lxd $USER**

**$ newgrp lxd**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XIII – Update do apt e instalação do snapd.**



**Imagem XIV – Configuração do user e criação do grupo.**

A seguir, criamos um diretório para o lxd, entramos no diretório, e iniciamos o lxd.

**Comandos:**

**$ sudo mkdir lxd-dire**

**$ cd lxd-dire**

**$ sudo lxd init**

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XV – Criação do diretório lxd-dire e init do lxd.**

**PASSO IMPORTANTE!!**

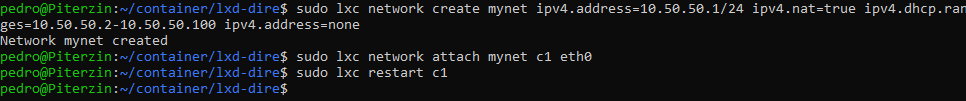
A seguir, se ao dar launch em seu container, obter falha, utilize os comando lxc list, se ao listar o container e observe se existe alguma rede conectada ao IPV4 ou IPV6, caso não esteja, rode os comandos a seguir para criar uma rede para o container.

**Comandos:**

**$sudo lxc network create mynet ipv4.addres=10.50.50.1/24 ipv4.nat=true ipv4.dhcp.ranges=10.50.50.2-10.50.50.100 ipv4.address=none**

**$ sudo lxc network attach mynet c1 eth0**

**$ sudo lxc restart c1**

**Imagem XVI – Criação da rede e conexão com o container.**

Assim, rodamos o launch do container c1, verificamos se a rede foi linkada adequadamente, utilziamos o start do container c1 e executamos o c1.

**Comandos:**

**$ lxc launch images:debian/12 c1**

**$ lxc list**

**$ lxc start c1**

**$ lxc exec c1 -- bash**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XVII – Criação da rede e conexão com o container.**

**Parte III (Docker)**

Para começar a parte de docker, devemos criar um novo diretório na nossa pasta container, criando alguns arquivos. OBS: Nos prints está errado, utilize "cd docker-dire" antes de criar os outros arquivos.

**Comandos:**

**$ sudo mkdir docker-dire**

**$ cd docker-dire**

**$ sudo touch Dockerfil**

**$ sudo touch app.py**

**$ sudo touch requirements.txt**

Texto

O conteúdo gerado por IA pode estar incorreto.

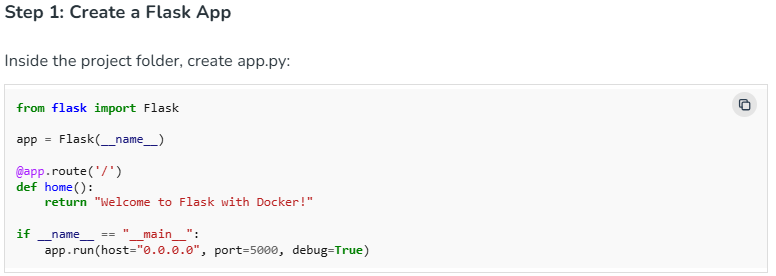
**Imagem XVIII – Criação do repositório e criação dos arquivos.**

A seguir, será necessário conseguir algumas informações, por meio da documentação do docker disponível na página <https://www.geeksforgeeks.org/devops/dockerize-your-flask-app/> . A primeira configuração que deve conseguir, são as congurações do app.py. Utilize o nano e cole o conteúdo da imagem XIX no app.py.

**Comandos:**

**$ sudo nano app.py**

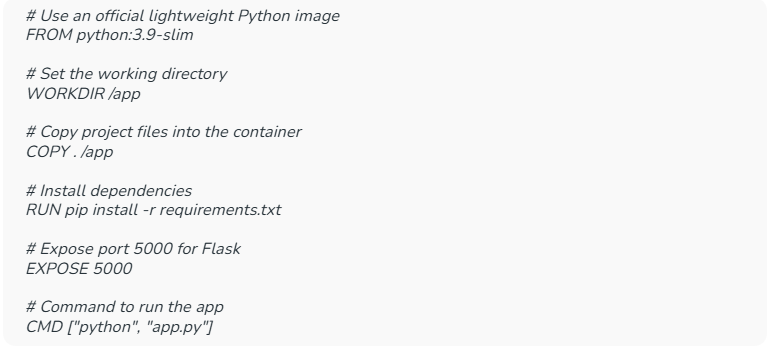
**$ code . (Se quiser abrir pelo VsCode)**

**Imagem XIX – Conteúdo do app.py.**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XX – Nano do app.py.**

Também na documentação do site disponibilizado, devemos conseguir algumas informações para o Dockerfile. Edite o arquivo do Dockerfile para colocar o seguinte conteúdo disponivel na imagem XXI.

**Imagem XXI – Conteúdo para o Dockerfile.**

O próximo passo para prosseguir é iniciar um ambiente virtual, e pegar os requirements para o docker. Para com todas as informações preenchidas poder finalmente rodar o build do docker. Com a build executada com sucesso, rode o docker.

**Comandos:**

**$ source venv/bin/activate**

**$ pip freeze > requirements.txt**

**$ pip install –r requirements.txt**

**$ docker build –t flask-docker-app .**

**$ docker run –p 5000:5000 flask-docker-app**

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXII – Comandos para configuração do docker.**

**Parte IV (Docker-rootless)**

A seguir, será necessário utilizar alguns comandos, pegos na documentação do docker-rootless disponível na página <https://docs.docker.com/engine/security/rootless/>. Assim, instalando algumas dependências e excluindo algumas configurações, que será necessário para execução do docker-rootless.

**Comandos:**

**$ sudo apt-get install –y dbus-user-session**

**$ sudo apt-get install –y uidmap**

**$ sudo apt-get install –y systemd-container**

**$ sudo systemctl disable docker.service**

**$ sudo systemctl disable docker.socket**

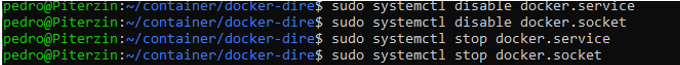
**$ sudo systemctl stop docker.service**

**$ sudo systemctl stop docker.socket**

**$ sudo systemctl disable –now docker.service docker.socket**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXII – Comandos para configuração do docker.**

**Imagem XXIII – Comandos para configuração do docker.**

**Imagem XXIV – Comandos para configuração do docker.**

Próximo passo para o processo, é instalar os pacotes de scripts do docker rootless

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXV – Comando para instalação de pacote.**

Texto branco sobre fundo preto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXVI – Comando para instalação de pacote.**

Agora, adicionamos os diretórios onde o Docker Rootless instalou os binários ao PATH. E fazemos com que comandos como docker e dockerd-rootless.sh sejam encontrados pelo shell sem precisar digitar o caminho completo. Também, configuramos o Docker Rootless para rodar como um serviço systemd de usuário.

**Comandos:**

**$ export PATH=$HOME/bin:$HOME/.local/bin:$PATH**

**$ dockerd-rootless-setuptool.sh install**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XXVII – Configuração do Docker rootless.**

A seguir, criamos e executamos um container, o comando também permite remover o container ao sair "--rm".

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXVIII – Comando para executar o container.**

**Parte V (Podman)**

Instala o Podman, que é um engine de contêineres compatível com Docker CLI, mas sem daemon e projetado para rodar rootless por padrão.

Permite criar e rodar contêineres OCI (mesmo formato que o Docker) sem precisar de dockerd. Também executamos o container utilizando o podman.

**Comandos:**

**$ sudo apt install podman**

**$ podman run –it –rm docker.io/library/python:latest**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXIX – Instalação do podman.**

Tela de computador com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.**Imagem XXX – Executar container com podman.**

**Parte VI (Git)**

utilizamos o codeberg e git para versionar os códigos, para isso, criamos uma conta no codeberg, criamos um repositório sem nenhuma das opções marcadas, somente com o nome. Após a criação do repositório, utilizamos os seguintes comandos para enviar os arquivos para o repositório, fazendo pasta por pasta

**Comandos:**

**$ touch README.md**

**$ sudo touch README.md**

**$ sudo git init**

**$ sudo git switch –c main**

**$ git config –global –add safe.directory (sua pasta, no meu caso é / home/pedro/container)**

**$ git add README.md**

**$ sudo git commit –m "first commit"**

**$ sudo git push –u origin main**

Agora, faça para cada pasta.

**RUNC**

**$ sudo git add runc-dire**

**$ sudo git commit –m "commit runc"**

**$ sudo git push –u origin main**

**LXD**

**$ sudo git add lxd-dire**

**$ sudo git commit –m "commit lxd"**

**$ sudo git push –u origin main**

**DOCKER**

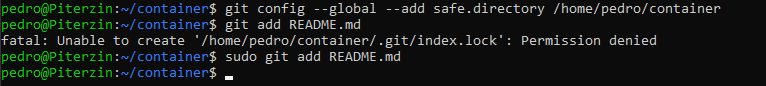
**$ sudo git add docker-dire**

**$ sudo git commit –m "commit docker"**

**$ sudo git push –u origin main**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXXI – Primeiro commit.**

**Imagem XXXII – Configurações e add**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXXIII – Commit.**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XXXIV – Push –u origin main.**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXXV – Add e commit runc.**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XXXVI – Push –u origin main.**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXXVII – Add e commit lxd.**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XXXVIII – Push –u origin main.**

Texto

O conteúdo gerado por IA pode estar incorreto.**Imagem XXXIX – Add e commit docker.**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Imagem XXXX – Push –u origin main.**

Para finalizar, falta apenas configurar o usuário.

**Comandos:**

**$ git config –global user.name "Piterzin"**

**$ git config –global user.email "manopar24@gmail.com"**

**$ sudo git push**

Texto

O conteúdo gerado por IA pode estar incorreto.

**Parte VII (Referências)**

* GEEKSFORGEEKS. Dockerize your Flask App. Disponível em: <https://www.geeksforgeeks.org/devops/dockerize-your-flask-app/>. Acesso em: 12 ago. 2025.
* DOCKER. Rootless mode. Disponível em: <https://docs.docker.com/engine/security/rootless/>. Acesso em: 12 ago. 2025.