

Containers - Trabalho TDS

Criação do repositório

1. Criar repositório remoto, pelo website <https://codeberg.org>
2. Em máquina local, instalar `git`, com o comando (em Fedora 38):

```
$ sudo dnf install git
```
3. E clonar o repositório remoto com comando

```
$ git clone https://codeberg.org/gabrielsantos46/trabalho-tds-containers.git
```
4. Após navegar até o diretório do repositório, podemos criar o arquivo `RELATORIO.md`, e gerar `RELATORIO.pdf` com o comando

```
$ pandoc RELATORIO.md -o RELATORIO.pdf
```
5. Podemos verificar que o repositório remoto está funcionando corretamente, realizando o primeiro `push`, com os seguintes comandos:

```
$ git add .  
$ git commit -m 'criacao de RELATORIO'  
$ git push
```

Container RunC

1. Instalar o pacote `RunC`:

```
$ sudo dnf install runc
```
2. Para configurar o `rootfs`, podemos utilizar a ferramenta `debootstrap`, que instala a base de um sistema Debian em um subdiretório de outro sistema já instalado

```
$ sudo dnf install debootstrap  
$ mkdir -p runc/rootfs && cd runc  
$ sudo debootstrap stable ./rootfs http://deb.debian.org/debian
```
3. Agora podemos gerar o arquivo de configuração `config.json`, executando:

```
[runc]$ runc spec --rootless
```

As configurações sobre o container são especificadas neste arquivo. Podemos observar, por exemplo, o caminho autodetectado do diretório `/`, o `hostname`, entre outros. Segue porção do arquivo `config.json`:

```
(...)  
"root": {  
  "path": "rootfs",  
  "readonly": true  
},  
"hostname": "runc",
```

```
"mounts": [
  {
    "destination": "/proc",
    "type": "proc",
    "source": "proc"
  },
  (...)
]
```

4. Finalmente, podemos executar o container, com o comando `runc run runc` (o nome do container foi automaticamente definido como o nome do diretório, `runc`). Como resultado, devemos obter um shell root dentro do container:

```
$ runc run runc
# test
```

5. O diretório `runc/rootfs` ocupa 301M, e consiste apenas de um sistema mínimo Debian, então não vamos adicioná-lo ao repositório. Para isso vamos voltar ao diretório raiz, e adicionar uma regra ao `.gitignore` deste repositório, assim como criar um arquivo shell que realiza a instalação do container.

```
$ cat >> .gitignore << EOF
runc/config.json
runc/rootfs/*
$ cat >> runc/create.sh << EOF
mkdir rootfs &&
sudo debootstrap stable rootfs http://deb.debian.org/debian &&
runc spec --rootless
```

Agora, para recriar o container RunC basta executar `sh create.sh` no diretório `runc`.

6. Ao executar `git add . && git status`, pode-se observar que apenas os arquivos `.gitignore` e `runc/create.sh` foram adicionados como arquivos novos

```
On branch main
Your branch is up to date with 'origin/main'.
```

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .gitignore
    modified:   RELATORIO.md
    modified:   RELATORIO.pdf
    new file:   runc/create.sh
```

Container LXC

Container Docker

1. Instalando pacote `docker`

```
$ sudo dnf install docker
```

2. Habilitando e inicializando o serviço `docker`, com `systemctl`

```
$ sudo systemctl enable docker
created symlink /etc/systemd/system/multi-user.target.wants/docker.service
→ /usr/lib/systemd/system/docker.service.
$ sudo systemctl start docker
$ sudo systemctl status docker
docker.service - Docker Application Container Engine
(...)
Active: active (running) since Fri 2023-12-08 15:10:43 -03; 2min 50s ago
(...)
```

3. Agora, podemos buscar e executar imagens do `dockerhub`

- Por exemplo, buscando uma imagem Debian:

```
$ sudo docker pull debian
$ sudo docker run debian
```

4. Podemos também criar uma imagem própria, a partir do `docker build`.

- Para isso, vamos criar um novo diretório, chamado `docker`, e um subdiretório, chamado `rootfs`

```
$ mkdir -p docker/rootfs && cd docker
```

- Utilizar, novamente, `debootstrap` para instalar um sistema Debian no diretório `rootfs`

```
$ sudo debootstrap stable rootfs http://deb.debian.org/debian
```

- E criar o arquivo `Dockerfile`

```
$ cat > Dockerfile << EOF
FROM scratch
ADD rootfs /
CMD ["/bin/sh"]
```

- A primeira linha, `FROM scratch` especifica que a imagem começa com sistema de arquivos vazio.
- A segunda, `ADD rootfs /` adiciona o diretório contendo um sistema de arquivos contendo Debian, montado no diretório `/` da imagem
- A terceira, `CMD ["/bin/sh"]` especifica que o comando em aspas deve ser executado na inicialização do container

- Agora podemos executar `docker build` para criar uma imagem com o título desejado:

```
$ sudo docker build -t tds-debian-build .
```

- E executar a imagem criada

```
$ sudo docker run -it tds-debian-build
# exit
```

Container Podman

1. Instalando pacote podman

```
$ sudo dnf install podman
```

Observamos que um pacote necessário para uso de modo rootless, `slirp4netns` já foi instalado e configurado:

```
(...)
```

```
Dependências resolvidas.
```

```
=====
Pacote                               Arq.   Versão           Repositório Tam.
=====
```

```
Instalando:
```

```
podman                               x86_64 5:4.7.2-1.fc38 updates      15 M
```

```
Instalando dependências:
```

```
(...)
```

```
Instalando dependências fracas:
```

```
(...)
```

```
slirp4netns                          x86_64 1.2.2-1.fc38   updates      47 k
```

```
(...)
```

2. Executando um container Podman qualquer

```
$ podman run --interactive --tty --rm debian /bin/sh
# exit
```

- `podman run` executa um container
- `--interactive` mantém o STDIN aberto, permitindo interação
- `--tty` aloca um pseudo TTY ao container, permitindo entrada de comandos
- `--rm` automaticamente remove o container ao sair
- `debian` busca uma imagem chamada “debian” em `docker.io/library` e a executa
- `/bin/sh` define qual shell executar ao iniciar o container