

REDES VIRTUALIZADAS COM DOCKER

Genileide Gabriella de Carvalho Ferreira¹, Samyta Ramiza Vicente da Silva¹, Maria Aparecida de Oliveira Varela¹, Natan Paulo Fernandes¹, Victor Carvalho Galvão de Freitas¹, Diego da Silva Pereira¹

*¹Laboratório de Pesquisa em Redes e Sistemas Computacionais (NOCS LAB),
Instituto Federal do Rio Grande do Norte, Parnamirim, Brasil
(gabriella.g@escolar.ifrn.edu.br)*

Resumo: A pesquisa demonstra a criação de redes virtualizadas com Docker, permitindo a comunicação segura entre containers. Foram comparados os métodos manual e Docker Compose, sendo este último mais eficiente por automatizar a configuração e reduzir erros. Os resultados confirmaram o isolamento da rede e a comunicação entre containers por meio do DNS interno do Docker, destacando vantagens como leveza, portabilidade e otimização de recursos.

Palavras-chave: Docker; Virtualização; Redes; Containers; Docker Compose.

INTRODUÇÃO

Serão apresentados conceitos como comunicação entre containers, isolamento de serviços e os principais tipos de redes do Docker, além de uma breve demonstração prática de criação e configuração de redes virtuais. O trabalho também destaca vantagens como flexibilidade, organização, escalabilidade e otimização de recursos em ambientes computacionais.

O Docker é uma plataforma de código aberto que automatiza o deploy de aplicações em containers. Foi criado pela DotCloud, empresa especialista em PaaS. Tendo como diferencial uma plataforma de deploy de aplicações sendo executada em um ambiente virtualizado de container (TURNBULL, 2014).

Containers são uma tecnologia de virtualização que permite replicar um ambiente computacional isolado, apto a executar aplicações de maneira consistente. Onde cada um deles é uma unidade virtual capaz de reunir todos os recursos necessários para o funcionamento de um serviço ou processo, podendo ser, códigos, bibliotecas, dependências e recursos de desenvolvimento (ROMERO, 2018).

O objetivo é a criação de uma rede virtual simples para que os containers possam conversar entre si de forma isolada e segura. O isolamento é garantido pelos recursos do Linux chamados **Namespaces** (isolamento de processos e rede) e **Cgroups** (limitação de hardware). A conexão é feita através de um driver bridge que funciona como um switch virtual conectando os containers. A maior vantagem da rede própria é que o Docker ativa um DNS

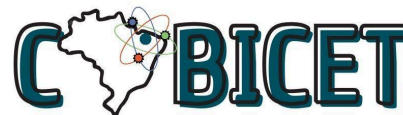
interno, permitindo o uso apenas do nome sem precisar do IP (GOMES, 2019).

MATERIAL E MÉTODOS

A metodologia utilizada nesta pesquisa teve como base a exploração da ferramenta Docker. O Docker é amplamente usado no gerenciamento de infraestrutura de aplicações de redes de computadores e temas correlatos. Seu manuseio é feito através do terminal de comando que permitiu a agilização dos processos de criação, manutenção e modificação dos serviços. Para o desenvolvimento, utilizaram-se o sistema operacional Windows 11 e a versão do Docker 29.4.3.

A criação da rede virtual foi feita utilizando-se de dois métodos, o manual e o Docker Compose, podendo haver um comparativo no nível de automação e organização da infraestrutura de ambos.

Na configuração manual, o primeiro passo foi a criação da rede, onde no terminal foi usado o comando: `docker network create rede-virtual`. A rede cria um ambiente de comunicação isolado. O segundo passo foi a criação da imagem, para isso foi preciso em uma pasta criar o arquivo Dockerfile com as instruções da imagem. E depois dentro da pasta onde está o arquivo executar o comando `docker image build -t minha-imagem .`, para construir a imagem. O último passo foi a criação dos 3 containers executando no terminal o comando `docker container run -d --name cont1 --net rede-virtual minha-imagem`, `docker container run -d --name cont2 --net rede-virtual minha-imagem` e `docker container run -d --name cont3 --net rede-virtual minha-imagem`. E por último foi feito o teste de



comunicação, fazendo o ping do container cont1 para o cont3 no terminal usando *docker container exec cont1 ping cont3*.

O segundo método foi utilizar o Docker Compose. O primeiro passo foi a criação, dentro de uma pasta, do arquivo *docker-compose.yml* com informações da imagem e dos containers. O próximo passo foi a criação da rede e dos containers executando no terminal, dentro da pasta do arquivo, o comando *docker-compose up -d*. E, por último, foi feito o teste de conexão executando o comando *docker-compose exec cont1 ping cont3*.

RESULTADOS E DISCUSSÃO

Na comunicação manual, ao usar o comando *docker network create*, foi possível observar o isolamento da rede virtual em relação à rede padrão Docker0 do host.

```
PS C:\WINDOWS\System32> docker network create rede-virtual
628902c67572f9355c6d843fadbb0cabc64b98d1f2149c8a1198fbc8bb898e
PS C:\WINDOWS\System32> docker network ls
NETWORK ID        NAME        DRIVER    SCOPE
df35867efbdc      bridge     bridge    local
9114d186a3ac      host       host      local
cf4e6959a757      none       none      local
628902c67572      rede-virtual bridge     local
PS C:\WINDOWS\System32> |
```

Figura 1. Criação e verificação de rede virtual no terminal.

```
FROM alpine
RUN apk update && apk add curl
CMD ["sh", "-c", "echo 'container ativo na rede' && sleep 3600"]
```

Figura 2. Criação da imagem no arquivo *dockerfile*.

```
PS C:\Users\genil\Documents\redeDocker> docker image build -t minha-imagem .
[*] Building 1.8s (6/6) FINISHED      docker:desktop-linux
=> [internal] load build definition from dockerfile      0.2s
=> [internal] load metadata for docker.io/library/alpine:latest      0.3s
=> [internal] load dockerignore      0.1s
=> [internal] transfer context: 20      0.0s
=> [1/2] FROM docker.io/library/alpine:latest@sha256:5b18f432ef3dab1      0.1s
=> resolve docker.io/library/alpine:latest@sha256:5b18f432ef3dab1      0.1s
=> CACHED [2/2] RUN apk update && apk add curl      0.8s
=> exporting manifest sha256:60f433c3d69e9d6e8a866e9d6d8ab5a959      0.7s
=> exporting layers      0.0s
=> exporting manifest sha256:c9f10885467142934251e625d4747242c5077      0.1s
=> exporting attestation manifest sha256:028a6f6d6e236c11a0a5033c      0.1s
=> exporting manifest list sha256:88a86d375af08a47c5931629d04d      0.1s
=> naming to docker.io/library/minha-imagem:latest      0.0s
=> unpacking to docker.io/library/minha-imagem:latest      0.3s
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/gfmxsdfp1q77gdxpnu5n6
```

Figura 3. execução do comando *build* para construção da imagem.

```
PS C:\WINDOWS\System32> docker container run -d --name cont1 --net rede-virtual minha-imagem
9ec89f81f938b14783b17882b1a6e9e2c5f4ab932b659b38f19af861b5a088
PS C:\WINDOWS\System32> docker container run -d --name cont2 --net rede-virtual minha-imagem
d15ef9a0156795c4b38c47f30b1b7bcb54660a8af8a35713ac6997adaa9
PS C:\WINDOWS\System32> docker container run -d --name cont3 --net rede-virtual minha-imagem
c6327f4f4183c0830f98d86549813a52be08865b4f4149ada8a298e019
```

Figura 4. Criação dos 3 containers.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c6327f4f418	minha-imagem	sh -c 'echo 'contai...''	7 seconds ago	Up 6 seconds		cont3
d15ef9a0156	minha-imagem	sh -c 'echo 'contai...''	13 seconds ago	Up 12 seconds		cont2
9ec89f81f93	minha-imagem	sh -c 'echo 'contai...''	19 seconds ago	Up 18 seconds		cont1

Figura 5. listagem dos containers ativos na rede.

Foi possível notar como a criação de uma rede utilizando-se do arquivo *docker-compose.yml*, simplificou o processo de subir múltiplos serviços de forma simultânea, fazendo com que erros de configuração manuais de rede fossem reduzidos.

```
services:
  cont1:
    image: alpine # Imagem base leve [4, 5]
    command: sh -c "echo 'container1 pronto' && sleep 3600"

  cont2:
    image: alpine
    command: sh -c "echo 'container2 pronto' && sleep 3600"

  cont3:
    image: alpine
    command: sh -c "echo 'container3 pronto' && sleep 3600"
```

Figura 6. Definição da estrutura Docker compose.

```
PS C:\Users\genil\Documents\redeDocker> notepad .\docker-compose.yml
PS C:\Users\genil\Documents\redeDocker> docker-compose up -d
[*] up 4/4
  Network rededocker_default Created 0.1s
  Container rededocker-cont3-1 Started 1.0s
  Container rededocker-cont2-1 Started 1.0s
  Container rededocker-cont1-1 Started 1.0s
PS C:\Users\genil\Documents\redeDocker> |
```

Figura 7. Execução automatizada.

Ao que diz respeito ao teste de ping por nome, foi obtido sucesso sem a necessidade de mapeamento do endereçamento ip manual, isso se deu pelo fato da utilização do DNS interno, característica tanto de redes manuais quanto no compose, que permite que os containers se comuniquem pelo nome.

```
PS C:\WINDOWS\System32> docker container exec -it cont1 ping cont3
PING cont3 (172.18.0.4): 56 data bytes
64 bytes from 172.18.0.4: seq=0 ttl=64 time=0.089 ms
64 bytes from 172.18.0.4: seq=1 ttl=64 time=11.940 ms
64 bytes from 172.18.0.4: seq=2 ttl=64 time=0.335 ms
64 bytes from 172.18.0.4: seq=3 ttl=64 time=0.243 ms
64 bytes from 172.18.0.4: seq=4 ttl=64 time=0.235 ms
64 bytes from 172.18.0.4: seq=5 ttl=64 time=0.237 ms
64 bytes from 172.18.0.4: seq=6 ttl=64 time=0.235 ms
64 bytes from 172.18.0.4: seq=7 ttl=64 time=0.222 ms
64 bytes from 172.18.0.4: seq=8 ttl=64 time=0.253 ms
64 bytes from 172.18.0.4: seq=9 ttl=64 time=0.232 ms
64 bytes from 172.18.0.4: seq=10 ttl=64 time=0.235 ms
64 bytes from 172.18.0.4: seq=11 ttl=64 time=0.159 ms
64 bytes from 172.18.0.4: seq=12 ttl=64 time=0.235 ms
64 bytes from 172.18.0.4: seq=13 ttl=64 time=0.155 ms
64 bytes from 172.18.0.4: seq=14 ttl=64 time=0.238 ms
64 bytes from 172.18.0.4: seq=15 ttl=64 time=0.379 ms
64 bytes from 172.18.0.4: seq=16 ttl=64 time=0.231 ms
64 bytes from 172.18.0.4: seq=17 ttl=64 time=0.234 ms
64 bytes from 172.18.0.4: seq=18 ttl=64 time=0.221 ms
64 bytes from 172.18.0.4: seq=19 ttl=64 time=0.240 ms
64 bytes from 172.18.0.4: seq=20 ttl=64 time=0.267 ms
64 bytes from 172.18.0.4: seq=21 ttl=64 time=0.234 ms
^C
--- cont3 ping statistics ---
22 packets transmitted, 22 packets received, 0% packet loss
round-trip min/avg/max = 0.089/0.766/11.940 ms
```

Figura 8. Teste de conectividade entre containers utilizando DNS interno.

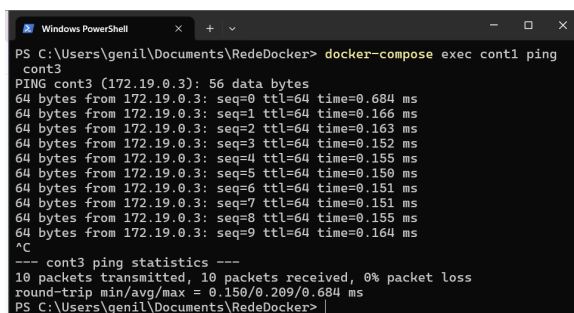
A screenshot of a Windows PowerShell terminal window. The prompt is 'PS C:\Users\genil\Documents\RedeDocker>'. The user has entered the command 'docker-compose exec cont1 ping cont3'. The output shows a series of ping requests to 172.19.0.3, each receiving 64 bytes with varying response times (e.g., 0.684 ms, 0.166 ms, 0.163 ms, 0.152 ms, 0.155 ms, 0.159 ms, 0.151 ms, 0.155 ms, 0.164 ms). After the pings, the user presses '^C' to stop the command. The terminal then displays '--- cont3 ping statistics ---', followed by '10 packets transmitted, 10 packets received, 0% packet loss', 'round-trip min/avg/max = 0.150/0.209/0.684 ms', and finally 'PS C:\Users\genil\Documents\RedeDocker> |'.

Figura 9. Teste de conectividade com método compose.

CONCLUSÃO

Por compartilhar o kernel do host, a virtualização por containers se torna mais leve do que as máquinas virtuais tradicionais. Além da utilização de redes customizadas com Docker Compose que garante portabilidade, facilidade de gerenciamento, permitindo que a aplicação seja replicada em diferentes ambientes de forma fiel.

Foi possível confirmar, como proposto na introdução, a criação de uma rede virtual simples para comunicação isolada e segura. A implementação demonstrou sucesso no isolamento através de *Namespaces* e na conectividade eficiente via driver *bridge*.

No comparativo entre os dois métodos, Manual e Docker Compose, foi possível evidenciar no método manual o isolamento em relação à rede padrão do host. Já o Docker Compose se mostrou superior na automação e organização, reduzindo erros de configuração humana ao subir múltiplos serviços simultaneamente.

As redes customizadas se mostraram vantajosas pela ativação do DNS interno, permitindo a comunicação entre os containers apenas pelo nome, simplificando o gerenciamento da infraestrutura.

Foi possível concluir que a leveza da virtualização por containers, em comparação com as máquinas virtuais tradicionais, traz mais portabilidade e a otimização de recursos necessários para ambientes de redes modernas.

AGRADECIMENTOS

Agradecemos ao professor Diego da Silva Pereira pelos conhecimentos compartilhados, pelo apoio e pelo incentivo durante o desenvolvimento deste trabalho. As orientações recebidas foram fundamentais para a realização desta pesquisa e

contribuíram significativamente para o nosso aprendizado e crescimento acadêmico.

E ao NOCS Lab e ao IFRN Parnamirim pelo apoio no desenvolvimento da pesquisa.

REFERÊNCIAS

- GOMES, Rafael. **Docker para desenvolvedores**. Leanpub, 2019.
- ROMERO, Daniel. **Containers com Docker: Do desenvolvimento à produção**. Casa do Código, São Paulo, 2018.
- TURNBULL, James. **The Docker Book**. Creative Commons, 2014.