

Análise Comparativa do Desempenho de Diferentes Drivers de Rede em Redes de Comunicação em Contêineres (Docker)

Michel de Souza Ramos¹, Mogleson de Lima Maciel¹

¹Universidade Federal do Ceará - Campus Quixadá (UFC)
CEP 63902-580 – Ceará – CE – Brasil

{michelsouza,moglesonlima}@alu.ufc.br,

Abstract. *This work aims to develop test scenarios using Docker container technology to perform performance evaluation tests of two different Docker Network Drivers, namely Driver Host and Driver Bridge, comparing some metrics, the main ones being bitrate and latency. As initial results, we obtained some data containing the aforementioned metrics through the Iperf3 and SockerPerf tools.*

Resumo. *Este trabalho tem como objetivo elaborar cenários de testes utilizando a tecnologia de contêineres Docker para realizar testes de avaliação de desempenho de dois diferentes Drivers de Rede Docker, sendo eles o Driver Host e o Driver Bridge, comparando algumas métricas, sendo as principais o bitrate e a latência. Como resultados iniciais, obtivemos alguns dados contendo as métricas supracitadas através das ferramentas Iperf3 e SockerPerf.*

1. Introdução

O Docker é uma plataforma projetada para facilitar o processo de criação, implantação e execução de aplicações em contêineres virtualizados em um Sistema Operacional comum, munida de um ecossistema de ferramentas aliadas. [Bhatia et al. 2017]. Essas aplicações são construídas e executadas em contêineres Docker, por meio de arquivos de imagens de aplicações. Um contêiner Docker é um *bucket* de software que oferece suporte à todas as dependências necessárias para que o software possa ser executado de forma independente. [Bhatia et al. 2017]. Em outras palavras, um contêiner Docker executa uma aplicação, com suas dependências, em um ambiente isolado.

Os contêineres são semelhantes às máquinas virtuais no que diz respeito aos serviços que fornecem, exceto que eles não vêm com sobrecarga de execução de um Kernel separado e de virtualização de todos os componentes de hardware. [Desai 2016]. Com o Docker, uma imagem de aplicação pode ser retirada de um repositório público ou privado, pronta para execução, consumindo menos recursos e isolada para que não interfira em outros ambientes. [Bhatia et al. 2017]. Podemos perceber que os contêineres apresentam-se como uma ótima alternativa à virtualização convencional, tendo como principais benefícios: a redução no consumo de recursos, a execução do contêiner em um ambiente isolado e a facilidade de implementação, tendo um vasto acervo de imagens de aplicações prontas.

A grande problemática abordada neste trabalho é a necessidade de utilizar programas e aplicações não compatíveis com determinados Sistemas Operacionais, determinados *frameworks*, bibliotecas e outros. Nem sempre se pode adaptar todo o código fonte de

um *framework* ou resolver todos os problemas de compatibilidade de um Sistema Operacional para que uma aplicação possa ser executada sem erros. Em vista disso, temos que um contêiner Docker oferece suporte a todas as dependências necessárias para a execução de uma aplicação, ou seja, o uso de contêineres oferece uma solução para o problema supracitado. Por exemplo, para executar uma aplicação em um contêiner Docker, basta que a máquina onde será executada possua o Docker instalado nela.

Dessa forma, este trabalho tem como objetivo analisar e comparar o desempenho de dois tipos de Drivers de Rede Docker, a fim de testar características de Redes de Contêineres Docker coletando métricas relevantes para efetuar essa análise. Os Drivers de Rede Docker abordados neste estudo serão os Drivers: Host e Bridge, tendo a finalidade de comparando o desempenho com a ausência e com a presença do Driver Bridge. A escolha das métricas será realizada em consonância ao objetivo deste trabalho, sendo posteriormente elencadas e justificadas.

2. Visão Geral do Docker

O Docker fornece a capacidade de empacotar e executar uma aplicação em um ambiente chamado contêiner, que é levemente isolado. Esse isolamento e a segurança permitem a execução de vários contêineres em um mesmo *host*, de forma simultânea. [Doc]. A plataforma Docker oferece ferramentas para criar, exportar e executar aplicações em contêineres Docker com facilidade.

Os contêineres são leves e possuem tudo o que é necessário para que a aplicação possa ser executada, ou seja, não é preciso depender do que está instalado no *host*. [Doc]. Não há necessidade do usuário se preocupar se o seu contêiner será enviado para uma máquina Ubuntu, uma máquina CentOS, ou qualquer outra; desde que a máquina possua o Docker instalado nela. [Miell and Sayers 2019]. Em um contêiner, a aplicação é executada de forma independente ao *host* e seu Sistema Operacional, visto que o próprio contêiner fornece todas as dependências necessárias para o funcionamento da aplicação, não sendo necessário se preocupar com compatibilidade.

O Docker torna mais simples e rápido o ciclo de vida do desenvolvimento, permitindo que os desenvolvedores trabalhem em ambientes padronizados com contêineres locais que possuem suas aplicações e serviços. [Doc].

2.1. Arquitetura do Docker

O Docker possui uma arquitetura cliente-servidor. O Cliente Docker se comunica com o daemon do Docker, sendo este que faz o trabalho pesado de construir, executar e distribuir os contêineres Docker. [...] O Cliente Docker e o daemon comunicam-se através de uma API REST, sobre *sockets* UNIX ou sobre uma interface de rede. [Doc]. Ademais, serão apresentados, brevemente, alguns dos principais componentes da arquitetura do Docker. A Figura 1 mostra a arquitetura do Docker.

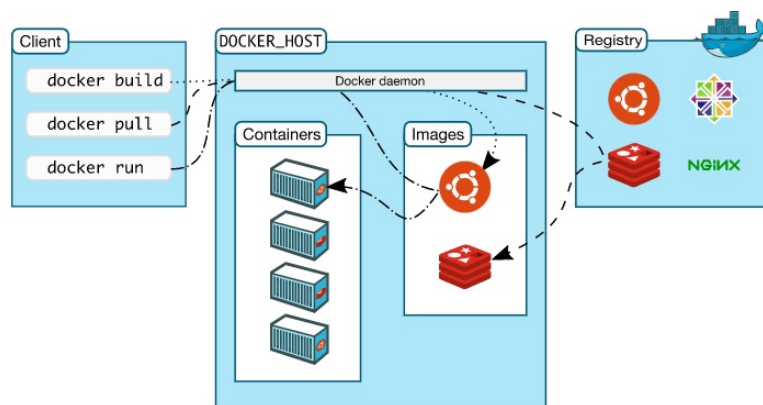


Figura 1. Arquitetura do Docker

Fonte: [Doc]

2.1.1. O Daemon Docker

O daemon do Docker, o dockerd, escuta as solicitações da API do Docker e realiza o gerenciamento de objetos Docker (imagens, contêineres, volumes e redes). Um daemon para pode comunicar-se com outros daemons para gerenciar os serviços Docker. [Doc].

2.1.2. O Cliente Docker

O Cliente Docker (docker) é a principal forma que os usuários do Docker podem usar para se comunicar com o Docker. [...] O comando docker utiliza a API do Docker e o cliente Docker pode se comunicar com mais de daemon. [Doc].

2.1.3. Objetos Docker

Ao se usar o Docker, se está criando e usando imagens, contêineres, redes, volumes, *plugins* e outros objetos. [Doc]. Neste seção, serão apresentados, brevemente, os objetos Imagem e Contêiner.

Imagem

Uma imagem é um modelo *read-only* contendo instruções para se criar um contêiner Docker. Em muitos casos, uma image baseia-se em outra, contendo apenas alguma personalização adicional. [Doc]. É possível criar uma imagem própria ou apenas utilizar uma imagem pré-existente publicada através de um registro.

Contêiner

Um contêiner nada mais é que uma instância executável de uma imagem. É possível criar, iniciar, parar ou excluir um contêiner usando a API Docker ou a CLI (*Command Line Interface*). Um contêiner pode ser conectado a uma ou mais redes, anexar armazenamento a ele ou até criar uma nova imagem baseada no seu estado atual. [Doc].

2.2. Drivers de Rede Docker

Nesta seção, abordaremos, brevemente, os Drivers de Rede Docker que serão utilizados neste trabalho. São eles, os drivers: Host e Bridge.

2.3. Driver Host

O Driver de rede Host é aconselhável para contêineres autônomos. O *host* representa a comunicação por meio de sockets, onde cada contêiner contém uma ou mais portas de servidor direcionadas a si mesmo. [Mentz et al. 2020]. Para contêineres autônomos, o *host* remove o isolamento de rede entre o contêiner e o hospedeiro do Docker, usando a rede do hospedeiro diretamente. [Doc].

O Driver de rede Host funciona apenas em hospedeiros Linux, e não possui suporte para o Docker Desktop para Mac, para Docker Desktop Windows ou Docker EE para Windows Server. [Doc].

2.3.1. Driver Bridge

O Driver de Rede padrão.[...] Redes *bridge* são geralmente utilizadas quando suas aplicações são executadas em contêineres autônomos que possuem necessidade de se comunicar. [Doc]. O Driver Bridge cria uma ponte virtual para conectar os contêineres a endereços IP válidos dentro do escopo local de uma sub-rede privada. [Mentz et al. 2020]. Em termos de rede, uma rede *bridge* funciona como um dispositivo da camada de enlace, que encaminha o tráfego entre os segmentos da rede. [Doc].

Basicamente, o Driver Bridge permite que contêineres Docker isolados em uma mesma máquina consigam se comunicar. Em termos de Docker, uma rede *bridge* utiliza uma ponte de software que permite que contêineres conectados à mesma rede *bridge* comuniquem-se entre si, enquanto provê isolamento de contêineres não conectados à essa rede. [Doc].

2.4. Cenário de Testes e Coleta de Métricas

O cenário de testes inicial consiste em utilizar um container rodando uma imagem do iperf3 como servidor, escutando na porta 5201 e um cliente do iperf3 que enviará uma requisição ao servidor.

Para iniciar o servidor, foi usado o seguinte comando (que cria um servidor iperf3 utilizando a imagem networkstatic/iperf3):

```
sudo docker run -it --rm --name=iperf3-server -p
5201:5201 networkstatic/iperf3 -s
```

Para iniciar o cliente, foi usado o seguinte comando (que além de enviar a requisição ao servidor, formata a saída em para json, a enviando para o arquivo test_iperf.json, permitindo a coleta das métricas):

```
sudo docker run -it --rm networkstatic/iperf3 -c
172.17.0.2 --json > test_iperf_container.json
```

No segundo cenário, com dois contêineres em modo de rede *bridge*, foi utilizada a ferramenta SockPerf para analisar, principalmente, a latência entre os contêineres conectados em modo *bridge*.

Dentro do primeiro contêiner foi executada a seguinte linha de comando (que iniciou o SockPerf como servidor, escutando na porta 12345 e utilizando tráfego tcp):

```
Serv -> sockperf sr --tcp -p 12345
```

Dentro do segundo contêiner, no lado cliente, foi executada a seguinte linha de comando para iniciar uma requisição ao servidor e salvando a saída em .csv:

```
sockperf ping-pong --tcp -i 172.20.0.3 -m 350 -t 100  
-p 12345 --full-log test.csv
```

Após isso, foi realizado um filtro no arquivo "test.csv" para diminuir a quantidade de linhas no arquivo para evitar uma grande demais de dados, usando o comando (salvando a saída no arquivo filtro.csv):

```
head -n 500 test.csv > filtro.csv
```

3. Justificativa das Métricas

As métricas que trabalharemos inicialmente são as métricas de taxa de bits (*bitrate*) e latência. Essas métricas foram escolhidas com base nos testes e nos trabalhos em que pesquisamos onde a taxa de transmissão, neste caso o *bitrate* e a latência são métricas abordadas e utilizadas em teste de avaliação de desempenho.

Além desse fator, as ferramentas que utilizamos, o Iperf3 e o SockPerf nos oferecem a capacidade de coletar essas métricas gerando o tráfego necessário para os nossos testes iniciais. Além do *bitrate* e da latência, coletamos outras métricas, porém as principais métricas a serem observadas são o *bitrate* e a latência. Ainda assim, as outras métricas poderão e deverão ser citadas e analisadas nas próximas etapas deste trabalho.

3.1. Resultados

Os resultados obtidos nos testes iniciais com o Sockperf podem ser observados, com saída csv, no seguinte link: https://github.com/MichelSouzaGit/analise_2022.2/blob/main/filtro.csv.

Os resultados obtidos nos testes iniciais com o Iperf3 podem ser observados, com saída em formato json, no seguinte link: https://github.com/MichelSouzaGit/analise_2022.2/blob/main/test_iperf_container.json.

4. Conclusão

Durante o planejamento dos cenários de testes se pretendia montar dois ambientes, visando testar mais de um driver de rede docker (*host*, *bridge*), para testar a comunicação entre dois contêineres em uma rede usando *bridge*, e como seria essa comunicação em um ambiente usando Driver Host. Isso tendo em vista que a latência tende a ser menor em uma rede usando o driver Host, pois será provido para o container o acesso direto a(as) interfaces de rede presente(s) na máquina onde o mesmo está em execução, ou seja, a comunicação do container não precisará passar por uma ponte, como ocorre quando se usa uma rede com o Driver Bridge.

Referências

- Docker documentation. <https://docs.docker.com/>. Acesso em: 25/09/2022.
- Bhatia, G., Choudhary, A., and Gupta, V. (2017). The road to docker: A survey. *International Journal of Advanced Research in Computer Science*, 8(8).
- Desai, P. R. (2016). A survey of performance comparison between virtual machines and containers. *Int. J. Comput. Sci. Eng*, 4(7):55–59.
- Mentz, L. L., Loch, W. J., and Koslovski, G. P. (2020). Comparative experimental analysis of docker container networking drivers. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–7. IEEE.
- Miell, I. and Sayers, A. (2019). *Docker in practice*. Simon and Schuster.