Análise Comparativa do Desempenho de Redes de Comunicação em Contêineres Docker

Michel de Souza Ramos[1], **Mogleson de Lima Maciel**[1]

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

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

Abstract. *Currently, there is a growing need to encapsulate high-performance applications in containers, making it possible to achieve better performances in scenarios where there are limitations on computational power, but the use of containers ends up implying some limitations when it comes to connectivity, communication between tasks executed in containers different. When working with containerized environments, network drivers are used as a form of abstraction to create communication networks for containers with specific characteristics. The purpose of this work is to carry out some experiments to evaluate the performance and compare the behavior of 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. *Atualmente é crescente a necessidade do encapsulamento de aplicações de alto desempenho em contêineres possibilitando alcançar melhores desempenhos em cenários onde se tem limitações de poder computacional, porém a utilização de contêineres acaba implicando em algumas limitações quando o assunto é conectividade, comunicação entre tarefas executadas em contêineres distintos. Quando se trabalha com ambientes conteinerizados, se tem a figura dos drivers de rede como uma forma de abstração para se criar redes de comunicação para contêineres com características específicas. O intuito deste trabalho é a realização de alguns experimentos para avaliar o desempenho, e comparar o comportamento de 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 atráves 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 outos 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 depedências 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.

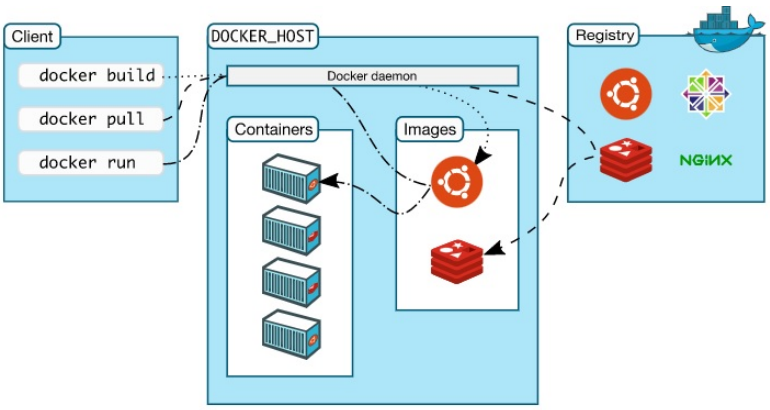

Figura 1. Arquitetura do Docker
Fonte: [Doc]

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 adiconal. [Doc]. É possível criar uma imagem própria ou apenas utlizar uma imagem pré-existente publicada atráves 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 utlizados 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 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 endereçõs 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 contêneires 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 definitivos para Driver Bridge podem ser observados no seguinte link, em JSON: `https://github.com/MichelSouzaGit/analise_2022.2/blob/main/Data/JSON/test_iperf_com_driver-bridge.json`.

Para os dados definitivos de Driver Bridge, em CSV, temos: `https://github.com/MichelSouzaGit/analise_2022.2/blob/main/Data/CSV/test_iperf_com_driver-bridge.csv`.

Os resultados obtidos nos testes definitivos para Driver Host podem ser observados no seguinte link, em JSON: `https://github.com/`

MichelSouzaGit/analise_2022.2/blob/main/Data/JSON/test_iperf_com_driver-host.json.

Para os dados definitivos de Driver Host, em CSV, temos: https://github.com/MichelSouzaGit/analise_2022.2/blob/main/Data/CSV/test_iperf_com_driver-host.csv.

4. Análise Descritiva

Nesta seção, realizaremos uma análise descritiva dos dados coletados anteriormente neste trabalho. É importante ressaltar que a Análise Exploratória bem como o Resumo de Dados e as Medidas-Resumo serão apresentadas para cada variável. Essa informações serão apresentadas separadamente por Driver utilizado. As variavéis foram selecionadas em acordo com os objetivos deste trabalho. Os valores aqui mostrados serão, posteriormente utilizados para tirarmos conclusões acerca dos dados.

4.1. Dados Coletados

Nesta seção, mostraremos as análises em relação aos dados coletados. No seguinte link: https://github.com/MichelSouzaGit/analise_2022.2/blob/main/Data/CSV/dataanalisys_hypothesistest.ipynb estão presentes todos os passos utilizados para a análise dos dados, desde as métricas de resumo até os testes de hipóteses. Destacaremos as principais variáveis que analisamos.

4.1.1. Variável TransmissionTime (Driver Bridge)

TransmissionTime é a variável do tempo que leva para a transmissão de cada fluxo. O tempo é medido em segundos com seis casas decimais, permitindo pegar os milisegundos, para obter um valor de tempo mais preciso. A Tabela 1 é referente aos dados analisados da variável TransmissionTime.

Tabela 1. TransmissionTime Bridge

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	1.0000009536743164
Media e Moda	1.0000000432795948; 0.9999970197677612
Soma Total	3600.0001558065414
Mínima e Máxima	0.998520016670227; 1.001471996307373

Fonte: Elaborada pelo autor

4.1.2. Variável PerSecondsBits (Driver Bridge)

PerSecondsBits é a variável que mede a quantidade de dados que é transmitida a cada segundo. Como o nome sugere, a medida foi coletada em bits. A Tabela 2 é referente aos dados analisados da variável PerSecondsBits.

Tabela 2. PerSecondsBits Bridge

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	243439951.03292263
Media e Moda	271666516.50354034; 242915032.11358
Soma Total	977999459412.7434
Mínima e Máxima	238744105.46136028; 909068891.4942428

Fonte: Elaborada pelo autor

4.1.3. Variável Latency (Driver Bridge)

Latency é a variável que apresenta a latência, em milissegundos, de cada transmissão. A Tabela 3 é referente aos dados analisados da variável SecondsStream.

Tabela 3. Latency Bridge

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	3521.0
Media e Moda	3451.1830555555557; 4095
Soma Total	12424259
Mínima e Máxima	779; 5961

Fonte: Elaborada pelo autor

4.1.4. Gráfico de Latência por Tempo de Transmissão (Driver Bridge)

Aqui, será apresentado um gráfico que tem como eixo x a variável TransmissionTime e como eixo y, a variável Latency. Por meio deste gráfico, poderemos visualizar melhor o comportamento dos dados coletados do Driver Bridge. A Figura 2 a seguir, apresenta o comportamento dos dados graficamente.

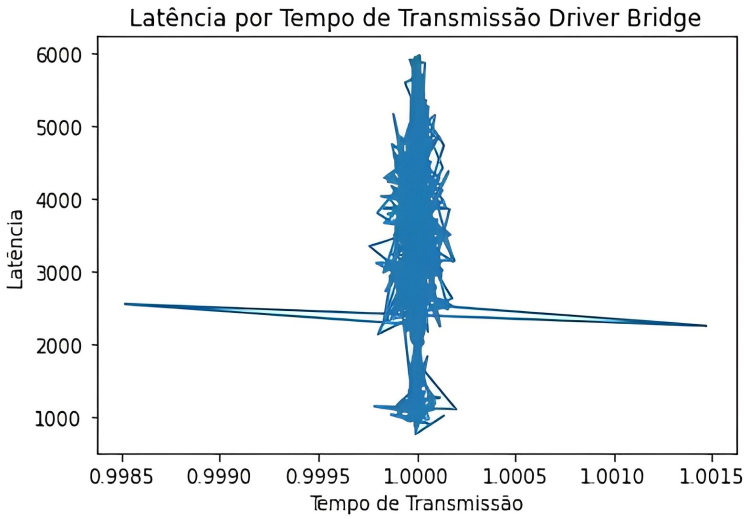

Figura 2. Gráfico de Linha: Latência por Tempo de Transmissão Driver Bridge
Fonte: Elaborada pelo autor

Podemos perceber que, nesse caso, a variação da latência tem uma concentração considerável de valores situados entre 3000 e 5000.

4.1.5. Variável TransmissionTime (Driver Host)

TransmissionTime é a variável do tempo que leva para a transmissão de cada fluxo. O tempo é medido em segundos com seis casas decimais, permitindo pegar os milisegundos, para obter um valor de tempo mais preciso. A Tabela 4 é referente aos dados analisados da variável TransmissionTime.

Tabela 4. TransmissionTime Host

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	1.0000009536743164
Media e Moda	1.0000000372197893; 1.0000029802322388
Soma Total	3600.0001339912415
Mínima e Máxima	0.997417986392975; 1.0025429725646973

Fonte: Elaborada pelo autor

4.1.6. Variável PerSecondsBits (Driver Host)

PerSecondsBits é a variável que mede a quantidade de dados que é transmitida a cada segundo. Como o nome sugere, a medida foi coletada em bits. A Tabela 5 é referente aos dados analisados da variável PerSecondsBits.

Tabela 5. PerSecondsBits Host

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	243956102.73479745
Media e Moda	271664281.44884044; 243437034.50110167
Soma Total	977991413215.8187
Mínima e Máxima	234578586.6650661; 941434485.693405

Fonte: Elaborada pelo autor

4.1.7. Variável Latency (Driver Host)

Latency é a variável que apresenta a latência, em milissegundos, de cada transmissão. A Tabela 6 é referente aos dados analisados da variável SecondsStream.

Tabela 6. Latency Host

Característica	Resultado
Variável (Tipo)	numeric
Número de observações ausentes	0
Mediana	2607.0
Media e Moda	2530.1144444444444; 2727
Soma Total	9108412
Mínima e Máxima	746; 5967

Fonte: Elaborada pelo autor

4.1.8. Gráfico de Latência por Tempo de Transmissão (Driver Host)

Aqui, será apresentado um gráfico que tem como eixo x a variável TransmissionTime e como eixo y, a variável Latency. Por meio deste gráfico, poderemos visualizar melhor o comportamento dos dados coletados do Driver Host. A Figura 3 a seguir, apresenta o comportamento dos dados graficamente.

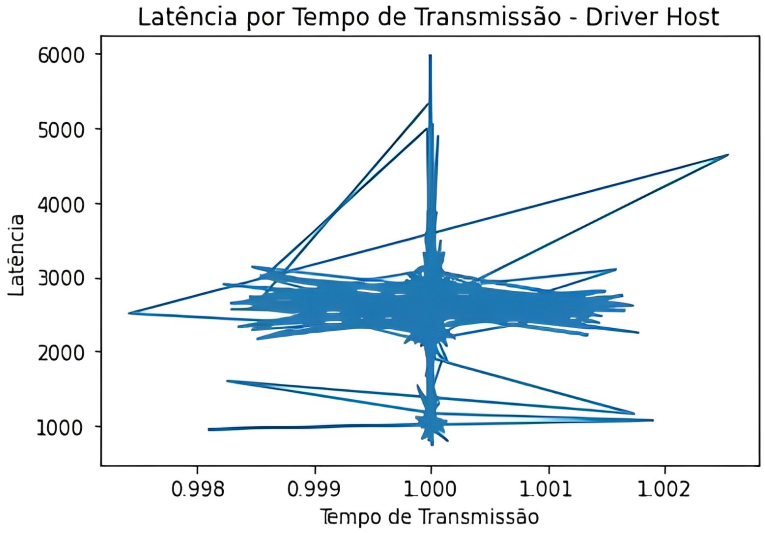

Figura 3. Gráfico de Linha: Latência por Tempo de Transmissão Driver Host
Fonte: Elaborada pelo autor

Podemos perceber que, nesse caso, ´por mais que a máxima de latência do Driver Host seja maior que a máxima do Driver Bridge, considerando a variação total da latência, e observando os valores presentes nas tabelas, o Driver Bridge levou vantagem sobre o Driver Host no quesito latência, que a métrica mais relevante para este trabalho.

5. Teste de Hipótese

Nesta seção, apresentaremos os testes de Hipóteses que elaboramos em consonância com o objetivo desde trabalho. A variável a ser avaliada no teste de hipótese será a variável de latência para ambos os cenários, Driver de Rede Bridge e Driver de Rede Host, pois essa é a variável considerada de maior importância para o nosso objetivo.

5.1. Hipótese Inicial

Em primeiro momento, tivemos que avaliar, a cerca do comportamento dos dados, que tipo de teste usaríamos: paramétrico ou não paramétrico. A partir disso, surgiram as seguintes hipóteses:

Para dados do Driver Bridge

H_0: Os dados do Driver Bridge seguem uma distribuição normal (I).

H_A: Os dados do Driver Bridge não seguem uma distribuição normal (II).

E para dados do Driver Host

H_0: Os dados do Driver Host seguem uma distribuição normal (I).

H_A: Os dados do Driver Host não seguem uma distribuição normal (II).

Em ambos os casos, precisamos descobrir se ambas as distribuições são normais ou não. Sabendo disso, teremos como escolher entre testes paramétricos (para distribuições normais) e testes não paramétricos (para distribuição não normais). A seguir, mostraremos duas formas diferentes as quais podemos atestar isso.

5.1.1. Teste Gráfico Q-Q

A primeira forma, foi atráves de uma representação gráfica utilizando a biblioteca statsmodels.api e a biblioteca matplotlib.pyplot do Python. Vale ressaltar, novamente que os passos utilizados para análise dos dados e teste de hipóteses encontram-se na seguinte URL `https://github.com/MichelSouzaGit/analise_2022.2/blob/main/Data/CSV/dataanalisys_hypothesistest.ipynb`.

”O gráfico Q-Q, ou gráfico de quantil a quantil, é um gráfico que testa a conformidade a dstribuição empiríca e a distribuição teórica fornecida”. [ref 2008]. As representações gráficas das distribuições de cada um dos dados de latência, para Host e Bridge, podem ser visualizadas nas Figurar 4 e 5.

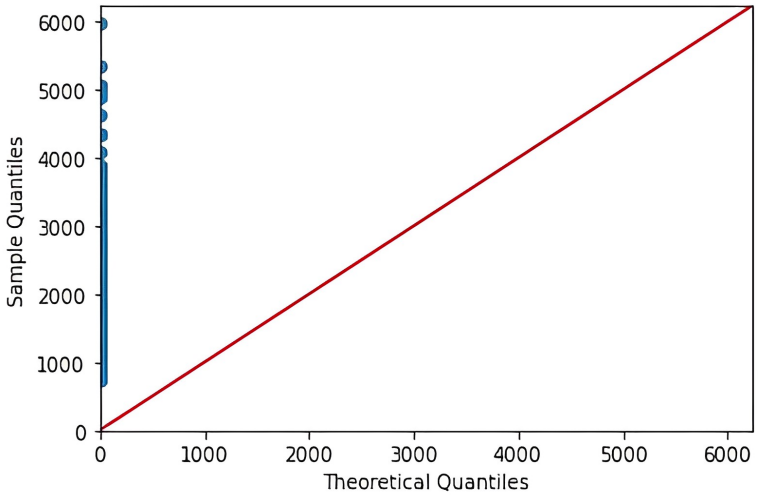

Figura 4. Gráfico Q-Q Host
Fonte: Elaborada pelo autor

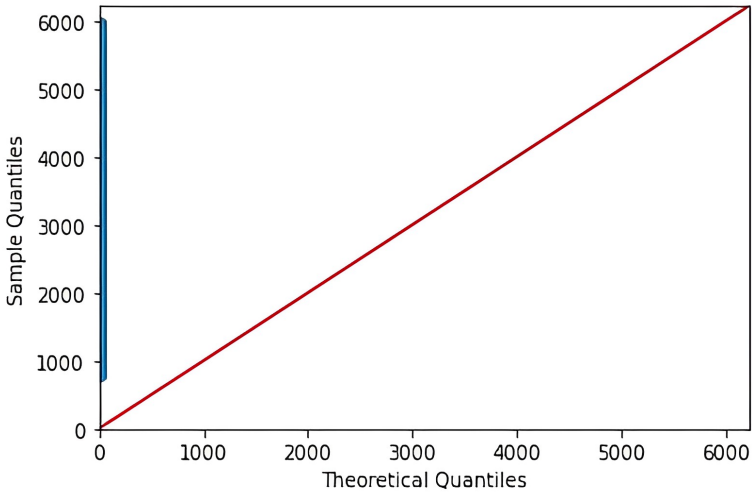

Figura 5. Gráfico Q-Q Bridge
Fonte: Elaborada pelo autor

Para ambos os gráficos, no eixo y, temos os *Sample Quanties* (Quantis de exemplo) e no eixo x os *Theorical Quanties*. O *Sample Quanties* são os valores dos nossos dados, e, como para cada distribuição, tanto para Host quanto para Bridge, temos um afastamento dos valores dos dados reais em relação ao valor teórico estimado dos dados para uma distribuição normal (*Theorical Quanties* temos que, em ambos os casos, nossas distribuições não seguem uma distribuição normal e, portanto, são dados não paramétricos.

5.1.2. Teste Shapiro-Wilk

Para corroborar essa conclusão, também realizamos o teste de *Shapiro-Wilk*. Conforme dito em [Sha] o teste de *Shapiro-Wilk* é utilizado para determinar se uma determinada amostra é de uma distribuição normal ou não.

Para o *P-value* obtido no teste de *Shapiro-Wilk* nos dados do driver Bridge foi igual a: $2.995123597537385e^{-18}$.

O *P-value* obtido no teste para o driver Host foi de: $1.401298464324817e^{-45}$.

Para ambos os casos, o *P-value* mostrou-se inferior a 0.05, portanto, segundo o teste de *Shapiro-Wilk*, temos evidências para comprovar que as dsitribuição são não normais e, consequentemente, não paramétricas.

Dessa forma, com base nessas evidências, temos resultados suficientemente satisfatórios para rejeitarmos a Hipótese Nula (I) e aceitarmos a Hipótese Alternativa (II) em ambos os casos, tal hipótese diz que nossos dados não seguem uma distribuição normal, ou seja, tratam-se de dados não paramétricos.

5.2. Hipótese com Teste Não Paramétrico

Conforne o teste de hipótese realizado na seção anterior demonstrou que os nossos dados observados na variável de latência para ambos os Drivers são não paramétricos, logo, escolhemos um teste não paramétrico a ser realizado.

5.2.1. Teste Mann-Whitney U

O teste selecionado foi o de *Mann-Whitney U*, pois foi julgado ser mais adequado para a comparação de nossas distribuições. O teste de *Mann-Whitney U*, segundo dito em [Man] é um teste para baseado em classificação para comparar dois grupos independentes de dados.

Formulamos, então, para este caso, as hipóteses:

H_0: Não há diferença estatística significante entre as medianas dos valores de Latência para os Drivers Host e Bridge. (I).

H_A: Há diferença estatística significante entre as medianas dos valores de Latência para os Drivers Host e Bridge. (II).

O resultado do teste de *Mann-Whitney U* nos retornou um valor aproximado para o *P-value* de 0.0. No Python, quando isso acontece, significa que o valor é tão pequeno

que pode chegar a ser desprezível e por isso é arredondado para zero.

O teste de *Mann-Whitney U* compara se há diferença significante entre os valores das medianas de duas distribuições independentes. Com o resultado inferior a 0.05, descartamos a hipótese nula e assumimos a hipótese alternativa. Isso significa que há difirença estatística significante entre os dados observados.

6. 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 maquina 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.

De acordo com os resultados obtidos na análise geral das variáveis de ambos os Drivers, podemos nota um grande número de valores parecidos e também algumas diferenças relevantes. No teste de hipótese, a partir da latência, podemos observar que houve uma diferença estatística signficante, conforme sugerido pelo teste de *Mann-Whitney U*. Dessa forma, podemos atestar um melhor desempenho do Driver de Rede Host, pois este apresentou menor latência total e média e mediana inferiores em relação ao Driver de Rede Bridge. Como o teste de *Mann-Whitney U* mostrou que a diferença estatística dos dados de latência para ambos os Drivers é significante, podemos corroborar a afirmação de que o desempenho do Driver Host foi superior na métrica observada no teste.

Referências

Docker documentation. https://docs.docker.com/. Acesso em: 25/09/2022.

How to perform a shapiro-wilk test in python. https://www.statology.org/shapiro-wilk-test-python/. Acesso em: 06/12/2022.

How to perform mann-whitney u test in python with scipy and pingouin. https://www.marsja.se/how-to-perform-mann-whitney-u-test-in-python-with-scipy-and-pingouin/. Acesso em: 05/12/2022.

(2008). *Q-Q Plot (Quantile to Quantile Plot)*, pages 437–439. Springer New York, New York, NY.

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.