Análise de desempenho de comunicação de contêineres Docker com os runtimes runC e Kata

Henrique Z. Cochak[1], **Charles C. Miers**[1]

[1]Departamento de Ciência da Computação (DCC)
Universidade do Estado de Santa Catarina (UDESC)

henrique.zc@edu.udesc.br,

charles.miers@udesc.br

Resumo. *O uso de contêineres propõe a execução de aplicações, suas bibliotecas e binários abstratos de modo virtualizado. A alteração do runtime, um dos módulos utilizados usado na execução de contêineres, é um dos fatores que podem causar variação de desempenho. Esse artigo apresenta os resultados iniciais de um comparativos entre runtimes runC e o Kata quanto ao desempenho de rede, focado em largura de banda e latência.*

1. Introdução

Há um crescimento gradual de computação em nuvem nos últimos anos, com aprimoramentos constantes de técnicas de virtualização e inclusão de novas abordagens [Microsoft 2020]. A adoção deste paradigma é economicamente viável devido aos conceitos de virtualização que permitem a criação de múltiplas aplicações na mesma máquina física dentro de máquinas virtuais (MVs) ou com contêineres.

Especificamente visando plataformas de gerenciamento de conteinerização, existe o Docker, que é uma plataforma aberta para desenvolvimento, envio e execução de aplicações conteinerizadas [Docker 2020]. O Docker, dentre todas as suas funções, contém uma em especial chamada runC *runtime*. Este *runtime* é um cliente de linha de comando que executa aplicativos empacotados, os contêineres, seguindo o formato dado pela *Open Container Initiative* (OCI), o que torna este cliente compatível com outras implementações no mesmo formato. O runC *runtime* é a parte mais importante, pois é este que instancia o processo do contêiner, a criação dos *namespaces* e associa toda a parte de comunicação de redes com o processo do contêiner. Por outro lado, o runC já possui alguns anos e com isso diversas novas necessidades surgiram nos contêineres. Uma das novas abordagens é *runtime* Kata que objetiva satisfazer primariamente requisitos de segurança.

O objetivo deste trabalho é de apresentar uma análise comparativa inicial do desempenho da comunicação de rede (largura de banda e latência) entre contêineres Docker ao utilizar dois tipos diferentes de implementação de *runtimes*: runC e Kata.

2. Contêineres Docker e Kata

Os contêineres são o resultado de uma tecnologia de virtualização em nível de sistema operacional (SO) com o princípio de executar qualquer tipo de aplicação independente de uma plataforma ou hardware específico. Este se torna um ambiente de execução pelo fato de agrupar sua aplicação, dependências, bibliotecas, arquivos de configuração e possíveis

binários em um único lugar, resultando em um ambiente abstrato de distribuições de SOs e sem a preocupação de diferenças de hardware subjacentes [Docker 2020].

Pelo fato de que o contêiner somente utiliza o que necessita para a sua aplicação executar, isto resulta em uma aplicação mais leve e uma implementação mais rápida, principalmente ao comparar-se com a tecnologia de MV. Buscando aprimorar ambas tecnologias, surgiram os Contêineres Kata, com a seu próprio *runtime*, dando a função de construir em tempo de execução MVs tão leves que são comparáveis a contêineres (Figura 1), adicionando a segurança fornecida de uma MV [Hunt 2020].

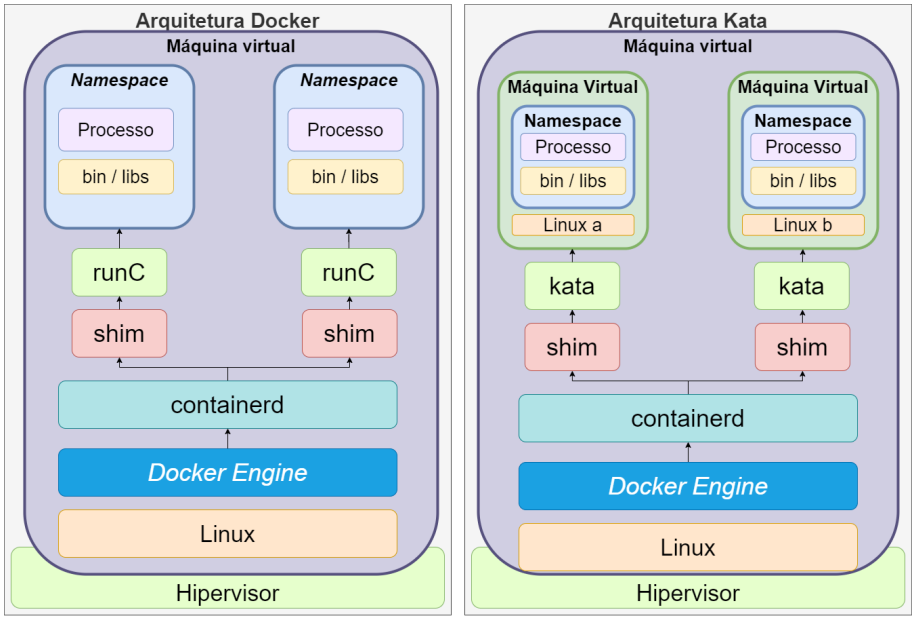

Figura 1. Comparativo de arquitetura entre Docker e Kata, ambos utilizando do Docker Engine como *daemon*

A diferença na escolha do *runtime*, em que encontra-se o runC para o Docker e o Kata *runtime* para o Kata, engloba principalmente a questão de segurança entre contêineres e MVs. Esta diferença sugere uma implementação diferenciada de *namespaces* de rede, na qual existe o isolamento feito pelo núcleo Linux dentro da *namespace* do hospedeiro para o runC, assim encontra-se uma única possível *namespace* de rede, isolada dentro de uma MV, para o Kata.

Um contêiner reside dentro do seu próprio *namespace* normalmente com a sua própria *namespace* de rede para garantir o isolamento de comunicação e dos seus recursos. Para que ocorra comunicação entre contêineres, é utilizado um par de interfaces *Virtual Ethernet* (vEth), de modo que é criada uma conexão entre diferentes *namespaces*. Esta abordagem não se aplica completamente ao Kata, devido a diferença existente de um contêiner que é gerenciado unicamente por um núcleo Linux compartilhado e uma MV que é gerenciada por um núcleo Linux dedicado.

3. Comparativo de comunicação entre as *runtimes*

Ao utilizar o Docker e suas funcionalidades, por padrão este utiliza da arquitetura chamada de *Container Network Model* (CNM) [Poulton 2019], a qual dita o formato de criação da rede e como é feita a sua conexão com os contêineres.

O mesmo é fortemente vinculado às funcionalidades de rede proporcionadas pelo núcleo Linux com escolhas feitas por seus desenvolvedores que acabam diferenciando as soluções (Tabela 1).

Tabela 1. Comparação de uso de tecnologias entre Docker e Kata

Critérios	**Docker**	**Kata**
Suporta tecnologias de rede do Linux?	Sim	Sim
Suporta vínculo de contêiner com o *namespace* de rede do hospedeiro?	Sim	Não
Cria uma máquina virtual para aumentar a segurança do contêiner?	Não	Sim
Utiliza de filtros de tráfego?	Não	Sim

O modelo CNM é padronizado pela OCI e como ambos o Docker e o contêineres Kata implementam este modelo, é possível utilizar o *runtime* Kata dentro do gerenciador do Docker, o *dockerd*. A CNM é fundamentalmente criado com o uso de três elementos chaves: a *sandbox*, vista como a configuração de uma pilha de rede, que abrange como funcionalidade o gerenciamento das interfaces do contêiner, tabela de roteamento e configurações de *Domain Name System* (DNS). O segundo elemento é o *endpoint* que faz a conexão da pilha de rede com uma interface vEth. Por fim tem-se o *network* que é a implementação virtual do padrão da *Institute of Electrical and Electronics Engineers* (IEEE) 802.1d *bridge*, se encaixando como um *switch* virtual [Poulton 2019].

Os contêineres Kata não implementam somente do padrão CNM pois a utilização de *endpoints* e pares vEth não é suficiente devido falta de habilidade de hipervisores ou MVs de lidar com interfaces vEth com a visão centralizada de *namespaces*. Para contornar a situação, o Kata *runtime* acaba empregando regras de filtro de tráfego de controle, resultando em um redirecionamento da rede entrando de uma interface, e.g., *eth0* para uma interface chamada *tap0* na qual agora é redirecionada para a interface *eth0* do contêiner que reside dentro da MV [Hunt 2020]. Neste sentido, a realização de um análise de desempenho de rede torna-se relevante.

4. Experimentação

Com o Docker versão 20.10.2 dentro de ambientes de MVs com GNU/Linux Ubuntu 18.04.5 LTS, é utilizada a ferramenta Iperf, para mensurar a taxa de transferência entre contêineres de diferentes *plugins* de *network*. Para analisar a latência, é mensurado o *round trip time* de pacotes de eco *Internet Control Message Protocol* (ICMP). Todos os testes com a ferramenta Iperf possuem o mesmo tamanho de janela TCP de 1 Mb.

O teste para o *runtime* runC consiste em criar um contêiner sendo visto como servidor pelo Iperf na rede *bridge*, um contêiner como cliente pelo Iperf nas redes *bridge*, *host* e *macvlan*, um serviço visto como servidor pelo Iperf no *swarm* e um serviço visto como cliente pelo Iperf no mesmo *swarm* (Figura 2). O teste para o *runtime* Kata é similar com uma única variação, como existe uma rejeição do Kata em relação ao vínculo de contêineres na rede *host*, não é efetuado os testes de latência e largura de banda na rede *host*.

A partir dos experimentos, é evidente que o *runtime* runC possui um melhor desempenho de comunicação de redes em relação ao *runtime* Kata (Figura 3). Um possível resultado da troca de desempenho é visto na necessidade de uma camada extra de encapsulamento com o uso de contêineres dentro de uma MV mínima, implicando em mais redirecionamentos de informações com o controle de tráfego e a comunicação do núcleo

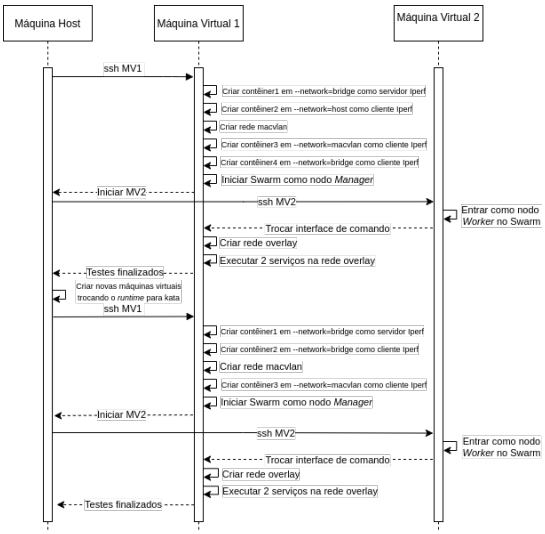

Figura 2. Sequência dos testes

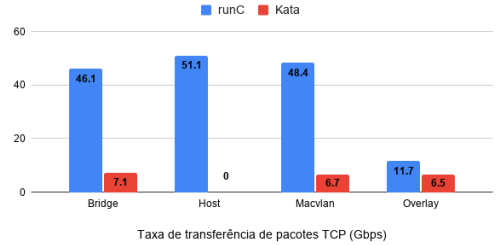

Figura 3. Resultados de largura de banda

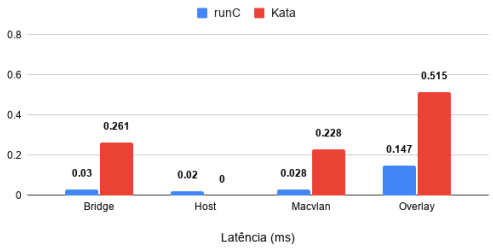

Figura 4. Resultados de latência

Linux. É relevante notar que existe uma rejeição de implementação vinda do Kata no vínculo de contêineres diretamente ao *namespace* de rede do hospedeiro (Figura 4), uma escolha dos desenvolvedores em prol da segurança, já que este tipo de conexão retira toda a camada de isolamento de um contêiner.

5. Considerações & Trabalhos Futuros

Tendo em conta que o objetivo é avaliar a capacidade de transmissão de pacotes TCP dentro do ambiente Docker com *runtimes* diferentes, existe uma troca de desempenho em favor da segurança oriunda da uma MV, na qual até o momento não aparenta ter sido otimizada o suficiente para obter um desempenho similar ao runC. Para trabalhos futuros, existem alguns direcionamentos: (i) a análise de ambas *runtimes* com outras ferramentas do mesmo perfil como o Iperf3 (similar ao Iperf contudo é desenvolvido e licenciado por outro grupo), (ii) a utilização de tamanho de janelas TCP diferentes e (iii) testar a vazão de dados também para pacotes UDP.

Agradecimentos: Os autores agradecem o apoio do LabP2D/UDESC e a FAPESC.

Referências

Docker (2020). Docker overview. https://docs.docker.com/get-started/overview/.

Hunt, J. O. D. (2020). Kata architecture. https://github.com/kata-containers/documentation/blob/master/design/architecture.md/.

Microsoft (2020). What is cloud computing. https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

Poulton, N. (2019). Docker deep dive. pages 64–72,206–226,236–252. JJNP Consulting Limited, 2018 edition.