UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
Autorizada pelo Decreto Federal nº 77.496 de 27/04/76
Recredenciamento pelo Decreto n°17.228 de 25/11/2016
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

XXIII SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DA UEFS
SEMANA NACIONAL DE CIENTÍFICA E TECNOLÓGICA – 2019

Estudo para definição da plataforma de virtualização com containers do LACAD

Antonio Raian de Lima Mendes[1]; Angelo Amâncio Duarte[2]

1. Bolsista PIBIC/FAPESB, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: raymendesjr2013@gmail.com
2. Orientador, Departamento de DETEC, Universidade Estadual de Feira de Santana, e-mail: angeloduarte@uefs.br

PALAVRAS-CHAVE: Virtualização, Container, Computação na Nuvem.

INTRODUÇÃO

A virtualização, criação de uma versão virtual de algum recurso computacional se difundiu a partir do conceito do *hypervisor*, um software que permite que executemos vários sistemas operacionais simultaneamente em uma única máquina. Com o *hypervisor* é possível criar máquinas virtuais, as quais possuem recursos virtuais isolados a partir de recursos físicos individuais [VERAS(2016)].

Com compartilhamento de máquina surge a necessidade de controlar o acesso dos usuários apenas ao que lhes fosse necessário. Esse conceito foi alavancado no ano de 2000 com a criação do *FreeBSD jail*, um sistema que criava “jaulas” para manter os aplicativos do usuário confinado em espaços apropriados [RED HAT(2018)]. Em pouco tempo, outros pesquisadores juntaram conceitos como: grupos de controles (*cgroups*), *systemd, namespaces, etc*, e desenvolveram uma tecnologia para acomodar todos os conhecimentos adquiridos, o *container*. Um *container* cria um novo nível de virtualização, e, como o nome já sugere, encapsula processos, os isola do sistema e fornece os arquivos necessários para o seu bom funcionamento.

Em 2008, a Canonical Ltd. e a Ubuntu começaram a elaboração de uma *Application Programming Interface* (API) para o gerenciamento de *containers*, o Linux Container (LXC). O LXC é uma interface de baixo nível que fornece contenção de recursos do *kernel* do Linux, possibilitando a gestão de *containers* e ambientes [CANONICAL(2018)]. O LXC fornece ferramentas, bibliotecas, associação de linguagens e modelos para melhorar a experiência dos usuários na utilização de *containers* [RED HAT(2018)].

Em 2009 entra em cena o DOCKER, que é um facilitador do uso de *containers* projetado pela Docker, Inc., configurando-se como combinação do LXC, só que com ferramentas ainda mais aprimoradas. O Docker tornou-se um dos projetos *open source* mais conhecido e mais utilizado no mercado para o gerenciamento e execução de *containers* [VITALINO & CASTRO(2016)].

Atualmente o Laboratório de Computação de Alto Desempenho (LaCAD) executa com diversos projetos que utilizam sistemas em nuvem de forma não homogênea, além de usar plataformas de desenvolvimento que requerem muito tempo para serem instaladas em novas máquinas. A partir dessa demanda, a ideia desse trabalho é avaliar as plataformas de virtualização usando *containers* disponíveis e experimentá-las no

contexto dos projetos do LaCAD, para estabelecer qual plataforma será adotada para os nossos trabalhos.

MATERIAL E MÉTODOS OU METODOLOGIA

O trabalho visa reproduzir a metodologia que foi indicada no artigo de Gupta e Gera (2016), que aborda uma comparação do desempenho das três plataformas de virtualização mais comuns no mercado, utilizando como medidas a largura de banda, a quantidade de acessos simultâneos e a velocidade de memória das plataformas. Diante disso selecionamos cinco aplicações para o *benchmark* neste trabalho:

1. *Dbench* [TRIDGELL(2008)]: Gera cargas de trabalho de entrada e saída em um sistema de arquivos local ou servidor. Podendo ser usado para mensurar a quantidade de acessos simultâneos que o sistema suporta até ficar sobrecarregado;
2. *GZIP Compression* [Free Software Foundation(2018)]: Mensura o desempenho de entrada e saída do sistema para compactar um arquivo de 2GB;
3. John the Ripper (*blowfish*) [Designer, S. (2014)]: Utilizada junto com o *BlowFish* para encriptografar e descriptografar dados;
4. *RAM Speed* [HOLLANDER(2018)]: Mensura a velocidade de comunicação de leitura e escrita entre a memória e suas caches;
5. *Stream* [McCALPIN(1991-2007)]: Mede a largura de banda estressando ao máximo os núcleos do processador.

Cada teste foi selecionado para avaliar uma área específica do desempenho das máquinas. O *Dbench* e o *GZIP* foram usados para análise do acesso ao sistema de arquivo, entrada e saída de cada *container*. O *Stream* e o *Blowfish* averíguam a velocidade dos núcleos do CPU, enquanto o *RAM Speed* testa a memória do sistema. Todos os testes foram encontrados no *Phoronix Test Suite* (PTS) [PHORONIX TEST SUIT(2019)], software que automatiza a execução dos *benchmarks*.

Os experimentos consistia em iniciar o computador apenas com o sistema operacional instalado. Em seguida instalar uma das plataformas (Docker ou LXD), criar um *container* com o PTS, e executar as baterias de testes. Para a plataforma Docker, a instalação e a execução do *container* foi bem mais simples, haja vista a existência de uma imagem do PTS disponível no Docker Hub (https://hub.docker.com/). Já na execução com o LXD foi necessária a criação e configuração manual do *container* com todas as dependências indispensáveis para a realização dos testes.

RESULTADOS E/OU DISCUSSÃO

1. Teste de Desempenho de CPU

Para testar a velocidade de processamento usou-se o *benchmark* John the Riper com o *Blowfish*, um algoritmo *open source* desenvolvido por Bruce Schneier(1993), que utiliza sistemas de blocos, chaves de 32 a 448bits e 16 iterações para criptografar e descriptografar *strings*. O aplicativo “JohntheRipper” é usado para calcular o tempo que a CPU leva para criptografar e descriptografar um conjunto de *strings*.

Na Figura 1 ilustra que o LXD é mais rápido, porém a diferença entre as plataformas é pequena, aproximadamente 0,29% (inferior a 20 criptografias por segundo em média). Com esse teste percebemos que o LXD é mais eficaz em explorar a CPU.

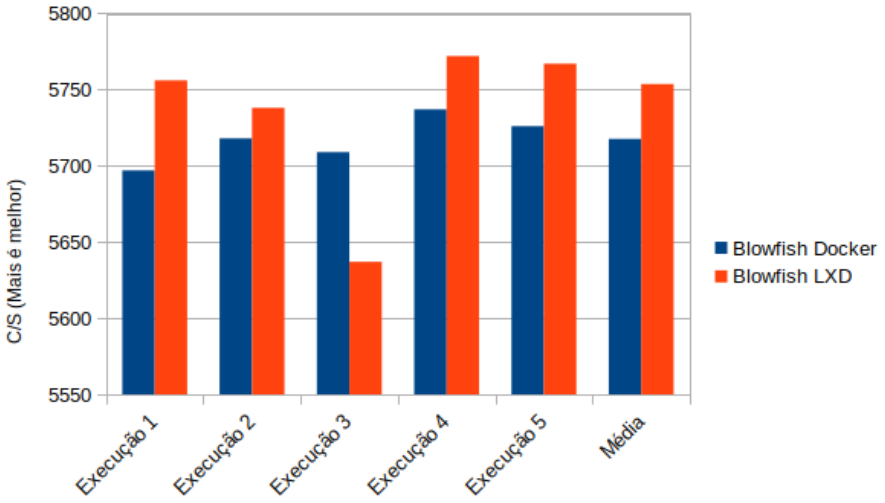

Figura 1: Execução do John the Ripper com o blowfish nas duas plataformas dispostas lado a lado;

2. Teste de Desempenho de Memória

O *Stream* é responsável por estimar a largura de banda (*bandwith*) em MB/s utilizada entre a memória do sistema e a CPU para realização de cálculos matemáticos. A Tabela 1 detalha a quantidade de bytes e de FLOPS (*Float Operations per Second*) para cada interação do Stream, em que **a(i)**, **b(i)** e **c(i)** são posições da memória para a iteração "i" e **q** é um escalar. As ações são de copiar o valor de *b* para *a* (Copy), somar *a* com *b* (Add), multiplicar *a* por *q* (Scale) e somar *a* com a multiplicação de *b* e *q* (Triad).

Tabela 1. Tabela de operações de STREAM

Nome	Função	Bytes/iteração	FLOPS/iteração
COPY	a(i) = b(i)	16	0
ADD	c(i) = a(i) + b(i)	16	1
SCALE	c(i) = q * a(i)	24	1
TRIAD	c(i) = a(i) + q * b(i)	24	2

Na Figura 2 nota-se o resultado da média de tempo das execuções para cada plataforma. A diferença entre as duas plataformas é de, aproximadamente, 1,5%, sendo o LXD o mais eficaz.

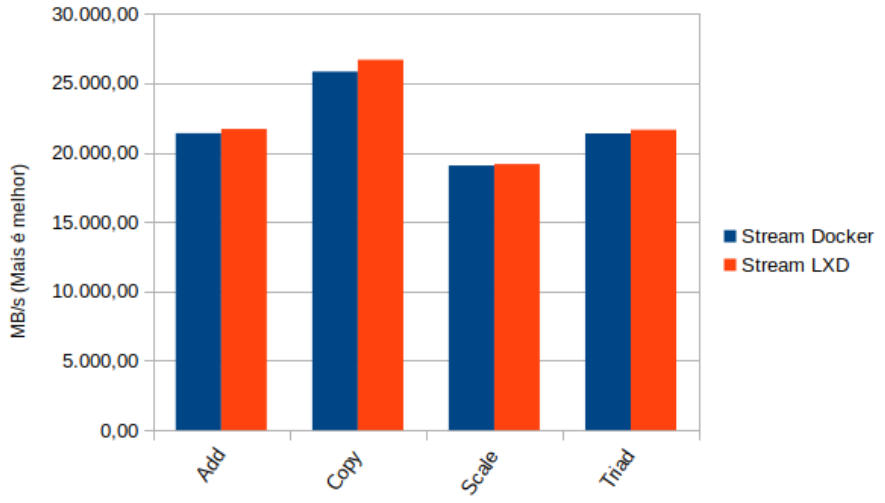

Figura 2: Execução do Stream nas duas plataformas colocadas lado a lado;

A execução do RAMSpeed com o LXD obteve uma eficácia maior, com quase 15% a mais de largura de banda. Dessa forma, conclui-se que a largura de banda das transações com o LXD é maior, e isso implica em maior fluxo de dados entre as memórias, impactando positivamente no desempenho do *container*.

3. Teste Desempenho de Entrada/Saída

Na avaliação de entrada e saída e de quantidade de acessos simultâneos foi usado o Dbench e GZIP Compress para a largura de banda em uma situação real. O Dbench coleta informações sobre o desempenho do *container* quando há acessos simultâneos ao sistema de arquivos, por usuários ou por algoritmos. O teste simula o acesso simultâneo de 6, 12, 128 e 256 clientes.

A Tabela 2 mostra uma grande diferença entre o LXD usando o sistema de arquivo ZFS e o Docker usando o “Device Mapper”. A diferença de desempenho é de quase 85%. Então, podemos dizer que o LXD seria a melhor opção ao usar o *container* para prover um serviço de acesso simultâneo.

Tabela 2. Tabela de Resultados Dbench

Clientes	**MB/s (mais é melhor**	
	Docker	**LXD**
6	25,09	133,67
12	40,58	230,33
128	106,67	321,33
256	96,77	314,33

Quanto ao GZIP, ele é utilizado para fazer a medição de largura de banda quando se trata de uma compressão de dados. Esse teste comprime um arquivo com mais de 2GB de tamanho e calcula o tempo necessário. Os resultados mostraram que a compressão de dados feita pelo Docker é mais eficiente, levando até 28 segundo a menos.

Analisando os dois testes chegamos à conclusão que o sistema de arquivos do ZFS é mais robusto e projetado para atender demandas enquanto o “Device Mapper” é mais eficaz para acesso unitário.

CONSIDERAÇÕES FINAIS

Com base nos resultados obtidos nos testes, o LXD apresentou maior desempenho para aplicações em que a largura de banda de memória é o requisito mais importante para o desempenho (aplicações *Memory Bound*). Não há qualquer diferença expressiva entre as duas plataformas para aplicativos em que a CPU é o item chave para o desempenho (aplicações *CPU Bound*).

O Docker se destaca na facilidade de uso e compatibilidade com Sistemas Operacionais populares no mercado, pois para configurá-lo é necessário, na maioria das vezes, executar apenas um *script* de instalação. O LXD, por sua vez, só é compatível com distribuições baseadas em Debian, e todo o processo de instalação tem que ser minunciosamente feito pelo usuário.

Além da facilidade de uso, o Docker, por ser uma plataforma *open source*, conta com vários colaboradores e um acervo repleto de imagens disponíveis para download. Isso compensa o fato de que o Docker apresenta menor desempenho para aplicações *Memory Bound*. À vista disso, o Docker é a escolha como plataforma de gestão de *containers* para o laboratório.

As demais atividades que seriam realizadas nos meses subsequentes, a saber: a elaboração de tutoriais, o treinamento dos integrantes do laboratório e a produção de uma palestra sobre as plataformas, não foram executadas dado o fim prematuro da pesquisa consequente ao início de atividades profissionais do orientando.

REFERÊNCIAS

McCALPIN, J. D. (1991-2007). Stream benchmark. Disponível em: http://www. cs.virginia.edu/stream/ref.html. Acesso em: 19 fev. 2019

TRIDGELL, R. A. (2008). Dbench. Disponível em: https://dbench.samba. org. Acesso em: 19 fev. 2019

Designer, S.(2014). Johntheripperpasswordcracker. Disponível em: https://www. openwall.com/john/. Acesso em: 19 fev. 2019.

VERAS, M. (2016). Virtualização. Tecnologia Central do Datacenter. 2.ed. Rio de Janeiro: Brasport, 2016

VITALINO, J.F.N e CASTRO, M.A.N (2016). Descomplicando o Docker. Rio de Janeiro: Brasport, 2016

GUPTA, Sapan; GERA, Deepanshu. A comparison of LXD, Docker and Virtual Machine. International Journal of Scientific & Engineering Research, v. 7, n. 9, 2016.

RED HAT, Inc. O que é um container Linux? Disponível em: https://www.redhat.com/ptbr/topics/containers/whats-a-linux-container. Acesso em 18 de Mar. 2018

CANONICAL(-1), Inc. What's LXC? Disponível em: https://linuxcontainers.org/lxc/introduction/. Acesso em: 23 de Mar. 2018

Free Software Foundation, I. (2018). Gzip compression. Disponível em: https:// openbenchmarking.org/test/pts/compress-gzip/. Acesso em: 19fev. 2019.

HOLLANDER, P. B. R. (2018). Ramspeed smp. Disponível em: https:// openbenchmarking.org/test/pts/ramspeed-1.4.1. Acesso em: 19fev. 2019.

PHORONIX TEST SUIT. Disponível em: https://www.phoronix-test-suite.com/ . Acesso em: 15 de Jan. de 2019