UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DOUGLAS BRAUNER

DOCB: UM BENCHMARK PARA AVALIAR AS ESTRATÉGIAS DE CONTÊINER E MÁQUINA VIRTUAL NA EXECUÇÃO DE APLICAÇÕES EM NUVEM COM ELASTICIDADE

São Leopoldo
2017

Douglas Brauner

DOCB: UM BENCHMARK PARA AVALIAR AS ESTRATÉGIAS DE CONTÊINER E MÁQUINA VIRTUAL NA EXECUÇÃO DE APLICAÇÕES EM NUVEM COM ELASTICIDADE

Artigo apresentado como requisito parcial para obtenção do título de Bacharel em Ciência da Computação pelo Curso de Ciência da Computação da Universidade do Vale do Rio dos Sinos – UNISINOS

Orientador: Prof.Dr. Rodrigo da Rosa Righi

São Leopoldo

2017

DOCB: UM BENCHMARK PARA AVALIAR AS ESTRATÉGIAS DE CONTÊINER E MÁQUINA VIRTUAL NA EXECUÇÃO DE APLICAÇÕES EM NUVEM COM ELASTICIDADE

Douglas Brauner*

Rodrigo da Rosa Righi**

Resumo: Uma das características mais importantes da computação em nuvem é a elasticidade de recursos, capacidade na qual o ambiente computacional pode aumentar ou diminuir os recursos demandados pelo usuário. Em ambientes de processamento de alto desempenho (PAD), as aplicações são encapsuladas em máquinas virtuais para garantir isolamento e permitir migração e replicação. Hoje em dia, além de máquinas virtuais, vem crescendo a adoção de Contêineres Docker, como uma alternativa à virtualização baseadas em hipervisor (VM), compostos de imagens leves e virtualizadas no nível de sistema operacional. Porém, por serem parte de uma tecnologia emergente, as aplicações PAD ainda utilizam o método de virtualização tradicional. Os estudos na literatura não oferecem uma comparação das tecnologias voltada para elasticidade em PAD.

Nesse contexto, este trabalho apresenta o DoCB (Docker Containers Benchmark), um benchmark de avaliação entre contêineres e máquinas virtuais, utilizando cada estratégia para a execução de aplicações paralelas e distribuídas em uma nuvem. O benchmark ainda mostra uma comparação de elasticidade e o impacto na execução em paralelo. Uma ferramenta desenvolvida no grupo de pesquisa, chamada AutoElastic, é utilizada para fazer o gerenciamento das execuções com contêineres e máquinas virtuais na nuvem, os dados gerados pelo AutoElastic são utilizados para compor e comparar os resultados de ambas tecnologias. Nos testes com uma aplicação de carga ascendente, a utilização de contêineres para virtualização apresentou um desempenho expressivamente maior, com um ganho de 20% no tempo de execução e 60% no custo (tempo x recursos) de processamento.

Palavras-chave: Computação em Nuvem. Elasticidade. AutoElastic. Contêiner. Docker.

1 INTRODUÇÃO

Uma das características mais importantes da computação em nuvem é a elasticidade de recursos (MELL; GRANCE, 2012; MAUCH; KUNZE; HILLENBRAND, 2013). Ou seja, a capacidade do ambiente computacional aumentar ou diminuir os recursos

* Aluno do curso de Ciência da Computação. Email: dbrauner@edu.unisinos.br

** Orientador, professor da Unisinos, pós-doutor pelo Korean Advanced Institute of Science and Technology (2013), doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul e pela Tecnische Universitaet Berlin (2009), Mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2005). Email: rrrighi@unisinos.br

demandados pelo usuário. Como recursos, podemos entender tudo aquilo que representa poder computacional, como CPU, memória, rede e largura de banda (KOMINOS et al., 2017). Uma das técnicas para se otimizar o desempenho de aplicações elásticas é a alocação dinâmica de recursos, o que permite o provisionamento de recursos quando a aplicação está necessitando, além da liberação de recursos adicionais, quando esta mesma está operando de forma moderada. A estratégia de elasticidade irá depender do objetivo do usuário, que por exemplo, em um cenário de aplicações de Processamento de Alto Desempenho (PAD), ou *High Performance Computing* (HPC), pode requisitar um aumento de poder computacional para executar uma determinada tarefa em um tempo menor. Por outro lado, se a execução não escala de forma linear e a aplicação não requer um tempo curto de processamento, a quantidade recursos pode ser reduzida, o que resultaria em um valor menor de *recursos x horas* para este caso, já que a questão é a economia de recursos.

O estado-da-arte atual mostra que uma das abordagens mais comuns de elasticidade é a replicação de máquinas virtuais (VMs), quando um determinado *threshold* é atingido, uma nova instância é requisitada e fornecida pelo gerenciador (R. RIGHI et al., 2016; TOSATTO; RUIU; ATTANASIO, 2015; HAN et al., 2012). Dessa forma, ambientes de PAD se beneficiam de utilização de tecnologias de virtualização para fornecer ambientes customizados, de acordo com as necessidades e permitir o compartilhamento de recursos. Entretanto, aplicações de alto desempenho somente serão capazes de se beneficiar da utilização de ambientes virtualizados se não houver um *overhead* substancial de desempenho: tanto no mapeamento de funções virtualizadas pelo hipervisor, quanto pelo tempo de entrega (*boot*) de uma máquina virtual (XAVIER et al., 2013). No geral, estudos realizados mostram que os métodos tradicionais de virtualização, como Xen, VMWare e KVM possuem um overhead significativo de desempenho para alto processamento e no tempo de entrega, o que dificulta a sua utilização em todas as aplicações de PAD (LI et al., 2017).

Algumas implementações recentes de virtualização baseada em contêineres, como Linux-Vserver, OpenVZ e Linux contêineres (LXC) oferecem uma camada leve de virtualização, com a promessa de desempenho muito próximo ao nativo, devido à sua arquitetura e virtualização no nível de sistema operacional (SO) (BERNSTEIN, 2014). Ao contrário de VMs, contêineres também oferecem uma solução sem isolamento,

onde contêineres num mesmo nó (máquina física), concorrem de igual para igual pela totalidade dos recursos da CPU, o que pode ser vantajoso, caso eles realizem tarefas de processamento intenso. Desta forma, podemos identificar 3 modalidades para estas duas técnicas de virtualização: máquina virtual com limite de CPU (Rígido), contêiner com limite de CPU (Rígido) e contêiner sem limite de CPU (Flexível). Todavia, não há trabalhos na literatura que explorem uma análise comparativa entre as 2 técnicas e 3 modalidades no contexto de PAD, tampouco no que tange o numero de unidades executadoras (VMs ou contêineres) por nó, considerando a variação da carga em tempo de execução.

Neste contexto, o presente artigo propõe o Docker Containers Benchmark (DoCB): uma avaliação de contêineres e máquinas virtuais, utilizando estratégias de cada alternativa, para a execução de aplicações paralelas e distribuídas em uma Nuvem. O *benchmarking* ainda mostra uma comparação de desempenho ao variar o número de unidades de processamento virtualizadas (contêiner ou VM), na execução em de uma aplicação em paralelo. O estudo mostra que contêineres conseguem tirar muito mais proveito da elasticidade na Nuvem, agilizando a entrega de recursos quando for solicitado, o *benchmarking* ainda mostra a maleabilidade de CPU na utilização vários contêineres em paralelo em uma mesma máquina, ou seja, a vantagem que se obtêm ao não definir limite de CPU por contêiner. O DoCB foi avaliado com a adaptação de um modelo chamado AutoElastic R. Righi et al. (2016), o qual foi utilizado para fazer o gerenciamento das execuções com contêineres e máquinas virtuais na nuvem, os dados gerados pelo AutoElastic são utilizados para compor e comparar os resultados de ambas tecnologias.

Este trabalho foi organizado em 8 seções. Nesta seção falamos do tema do trabalho e introduzimos o assunto abordado. Na sequência, iremos apresentar uma revisão dos conceitos por trás das tecnologias utilizadas e revisaremos o estado-da-arte. Na seção 3 é feito um estudo de trabalhos relacionados ao tema deste artigo. Na seção 4 será apresentado o *benchmarking* criado, com detalhes do modelo definido, seguindo pela metodologia empregada na execução, na seção 5. Teremos então, na seção 6 uma análise dos resultados obtidos utilizando os critérios definidos. Finalmente, a seção 7 é destinada às considerações finais do trabalho, bem como oportunidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo irá abordar as principais definições relacionadas ao conceito de Computação em Nuvem, levando em consideração o estado atual na área. Serão apresentados também conceitos necessários ao entendimento das motivações do projeto que levaram à definição do benchmark DoCB. Primeiramente, serão apresentados os conceitos e características de Computação em Nuvem, seguido posteriormente, dos conceitos relevantes ao entendimento da tecnologia de contêineres e a ferramenta mais utilizada atualmente no mercado, o Docker. Para finalizar, será traçado um comparativo entre as alternativas de virtualização: por máquinas virtuais ou contêineres.

2.1 Computação em Nuvem

Segundo a definição do National Institute of Standards and Technologies (NIST) (MELL; GRANCE, 2012), a computação em nuvem é um modelo que permite o acesso de recursos computacionais configuráveis e compartilhados, de forma conveniente, ubíqua e sob demanda, capazes de ser provisionados e entregues com baixo custo de gerenciamento e sem a intervenção de um provedor de serviços. Tais recursos podem ser: redes, servidores, serviços, armazenamento, aplicações, etc. Tais recursos são tipicamente fornecidos no modelo *pay-as-you-go*, ou seja, você paga de acordo com a demanda de recursos solicitada (SULEIMAN et al., 2012). Sendo assim, a computação em nuvem se utiliza de mecanismos para escalar estes recursos conforme necessidade, através de algoritmos que irão trabalhar no balanceamento de carga rapidamente para diminuir o desperdício de recursos computacionais.

Na computação em nuvem, existem cinco características essenciais que ajudam a defini-la, que são a clara distinção com outros paradigma, a elasticidade rápida dos recursos, medição de serviço, serviço sob demanda, pool de recursos e o amplo acesso ao meio (SOUSA; MOREIRA; MACHADO, 2010). Em relação à elasticidade, Taurion (2009) define que elasticidade é a capacidade do ambiente computacional da nuvem aumentar ou diminuir de forma automática os recursos computacionais demandados e provisionados para cada usuário. Diferente da escalabilidade, que se diz respeito à quantidade de usuários que ela pode manter conectados ao mesmo tempo, sendo o

limite de escalabilidade o ponto em que esta não pode suportar mais usuários conectados, sem apresentar a mesma eficiência (ROUMELIOTIS, 2012).

A elasticidade é a escalabilidade em duas direções: tanto cresce quanto diminui a capacidade ofertada. Os dois termos podem ser confundidos, já que dizem respeito à adaptação de recursos conforme demanda, porém a escalabilidade é relativa à capacidade manter o desempenho conforme a demanda, já a elasticidade é a capacidade de fazer os recursos se adaptem à carga, seja para aumentar ou diminuir. Iremos adotar a definição de Coutinho et al. (2013) que diz que a elasticidade é a *"Capacidade de adicionar e remover recursos de forma automática de acordo com a carga de trabalho sem interrupções e utilizando os recursos de forma otimizada".*

2.2 Docker

O Docker[1], apesar de inicialmente utilizar o LXC como provedor de *runtime*, criou uma implementação própria de ferramenta para criação de contêineres, procurando resolver algumas limitações da implementação de LXC, como segurança e simplificação, a ferramenta é atualmente a forma mais popular de gerenciar contêineres (PAHL, 2015). Segundo (NICKOLOFF, 2016), Docker pode ser descrito como um conjunto de ferramentas e serviço que facilitam a criação e manipulação de contêineres dentro de um SO baseado em UNIX. Alguns dos conceitos chave para compreender a tecnologia são descritos abaixo (DOCKER, 2016):

- Contêiner: A palavra contêiner *container* se refere a uma técnica de isolamento de recursos dentro de um SO baseado em UNIX, utilizando uma série de ferramentas que trabalham nativamente, mas sem a sobrecarga de rodar um *kernel* separado e sem fazer simulação de hardware, através de utilização de *namespaces*, e *control groups* (GRABER, 2014).

- Cgroups: Cgroups (*control groups*) é uma configuração que faz parte do subsistema de *kernel* de sistemas baseado em UNIX, que fornece controle sobre recursos do sistema, como CPU, memória, rede, etc (NICKOLOFF, 2016).

- Chroot: (*change root*) é um comando Linux para mudar o diretório raiz do pro-

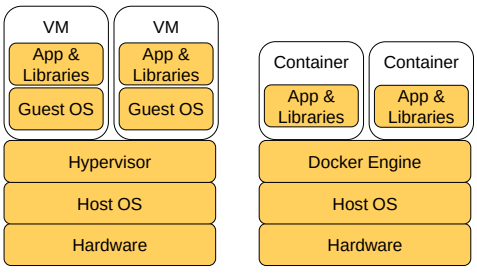
[1] https://www.docker.com/

cesso corrente e seus processos aninhados para um novo diretório. Alguns contêineres utilizam este comando para isolar e compartilhar o sistema de arquivos entre os ambientes contêinerizados (DUA; RAJA; KAKADIA, 2014).

2.3 Virtualização vs Contêinerização

Uma camada de virtualização pode ajudar a isolar um ambiente compartilhado de computadores. Em uma Nuvem de PAD, normalmente os recursos são compartilhados entre mais usuários, e isto faz com que possa ocorrer problemas nesse compartilhamento (XAVIER et al., 2013). É possível utilizar uma nuvem sem virtualização, o que trará vantagens em relação ao desempenho e evita o *overhead* nos recursos, porém, isto também significa falta de flexibilidade, e dificuldade de utilização, pois os recursos não são compartilhados entres o nós (KOMINOS et al., 2017). Um hipervisor (*hypervisor*), ou gerenciador de máquinas virtuais, é um programa que roda no sistema operacional do hospedeiro, fornecendo os recursos de hardware deste para uma série de máquinas virtuais, neste caso, estas máquinas virtuais compartilham o mesmo hardware. Como mostrado na Figura 1, o hipervisor é responsável por fornecer o acesso aos recursos para cada sistema operacional dentro das máquinas virtuais, mas com o custo deste gerenciamento como processamento (ZHANG et al., 2016).

Figura 1: Máquina Virtual vs Docker



Fonte: Elaborado pelo autor

O termo *contêinerização* é a forma popular para se referenciar à virtualização baseada em contêiner, que permite o isolamento de determinados softwares, que são executados dentro do mesmo kernel no sistema operacional Linux. Diferentemente de hipervisores, o Docker não adiciona uma camada virtualização, na qual precisaria carregar um sistema operacional convidado (ZHANG et al., 2016). Os contêineres

possuem a vantagem de utilizar diretamente os recursos do hospedeiro e não necessitar de tempo de *boot*, além de não possuir o *overhead* de processamento ao traduzir os comandos do sistema operacional convidado para o hospedeiro (DUA; RAJA; KAKADIA, 2014).

3 TRABALHOS RELACIONADOS

Nesta seção, será visto um estudo de trabalhos que trazem soluções de elasticidade para aplicações de alto desempenho, bem como estudos comparativos entre a utilização de máquinas virtuais e contêineres, considerados o estado-da-arte na atualidade. Este estudo tem por objetivo confrontar os trabalhos e identificar as lacunas e possíveis melhorias em ambientes elásticos. A seleção de trabalhos relacionados, bem como outras referências, foi realizada através de uma pesquisa em diversas bases de dados de artigos científicos de Ciências Exatas, como o portal da *IEEE Xplore*, *ACM Digital Library* e *Springer Link*, além da bases de dados nacionais, o Portal de Periódicos CAPES/MEC.

Para a realização das pesquisas nessas bases de dados, foram utilizadas buscas com estas principais palavras-chave: *docker virtual machine, container high performance computing, scalability elasticity, hpc application container, virtualization contêineres, cloud computing elasticity*. O resultado dessas pesquisas foi analisado com o intuito de encontrar trabalhos que explorassem a utilização de contêineres em ambientes de computação em nuvem, bem como as alternativas e implementações de elasticidade em aplicações PAD. Sendo assim, verificou-se os 30 primeiros resultados de cada pesquisa e foram descartados resultados focados em áreas muito específicas, ou que não contivessem dados de avaliação de desempenho.

3.1 Análise do Estado-da-Arte

O AutoElastic, segundo a definição de R. Righi et al. (2016), age como um *middleware* permitindo que aplicações PAD iterativas obtenham vantagem do provisionamento dinâmico de recursos de uma infraestrutura de nuvem sem a necessidade de modificações no código fonte. O AutoElastic possui um protótipo executado pelo autor

na plataforma de nuvem OpenNebula e obteve resultados de ganho de desempenho de até 59% na execução de uma aplicação de integração numérica *CPU-Bound*, quando comparada com outras soluções de elasticidade.

Chung et al. (2016) fazem um estudo comparativo de máquinas virtuais e contêineres, além de propor um modelo de *deploy* de aplicações distribuídas em contêineres Docker. Para verificar o modelo proposto, foram realizados testes utilizando duas aplicações PAD que requerem grande capacidade computacional, Graph500 e Linpack (HPL). Para manter um base de comparação, foram realizados execuções com processamento nativo, sem adição de camada de virtualização. Os testes executados mediam os custos computacionais dos processos, considerando diversas combinações de instâncias de máquinas virtuais e instâncias de contêineres, sem levar em consideração a elasticidade.

Dua, Raja e Kakadia (2014) apresentam um estudo sobre como provedores de PaaS (*Platform as a Service*)estão utilizando contêineres para encapsular aplicações. O estudo indaga a atual adoção de plataformas baseadas em contêineres e explora diversas implementações de contêineres, entre elas: Linux contêineres, Docker, Warden Container, lmctfy e OpenVZ. A análise foi feita baseada em como cada uma das tecnologias lidam com processos, sistema de arquivos e isolamento de namespaces, dando uma atenção especial à características únicas de cada implementação. Por fim, o trabalho busca fazer uma análise dos fatores que afetam a adoção de contêineres e possíveis funcionalidades que estão em falta para uma nova geração de PaaS.

O estudo de Xavier et al. (2013) aborda a utilização de tecnologias de virtualização para aplicações de alto desempenho, alegando que tais tecnologias foram tradicionalmente evitadas ao longo dos anos por sua adição de camadas que comprometem desempenho, o que é crucial para tais aplicações. Porém, como o advento de implementações de virtualização baseada em contêineres, o autor indaga que é possível se obter uma sobrecarga muito pequena com a utilização de tecnologias como Linux VServer, OpenVZ e Linux contêineres (LXC), chegando a um desempenho quase nativo. Para comparação, foi utilizado o Xen, um modelo tradicional de virtualização. Os testes utilizaram programas de avaliação de desempenho de supercomputadores fornecidos pela NASA (DUNBAR, 2016). O LXC executou os programas de teste com praticamente o mesmo potencial que a própria máquina, enquanto a implementação

utilizando máquina virtual (XEN) obteve uma sobrecarga de aproximadamente 4.3%.

O trabalho de Zhang et al. (2016) aborda a sobrecarga que soluções de hipervisor possuem em relação aos dispositivos de I/O virtualizados. O trabalho apresenta SRV-IOV (*Single Root I/O Virtualization*) para permitir o compartilhamento entre conexões de alto desempenho, como InfiniBand, além de introduzir testes utilizando contêineres, buscando atingir um desempenho mais próximo do nativo. Nos testes, utilizando aplicações de *benchmark* com MPI, as avaliações identificaram que VM com passagem PCI é mais eficiente que VM com SR-IOV habilitado, entretanto, contêineres possuem mais eficiência ao compartilhar recursos de I/O. Em comparação ao desempenho nativo, os testes identificaram uma sobrecarga para contêineres de no máximo 9% para aplicações PAD.

Adufu et al. (2015) conduziram testes para demonstrar que, durante a execução de aplicações científicas em ambientes PAD, o tempo médio de processamento em um sistema com virtualização baseada em contêineres é menor do que o tempo em um sistema baseado em hipervisores, isto devido ao tempo de *start-up*. Para gerar os resultados de comparação, foi utilizado a ferramenta *autodock3*, um programa de simulação de modelagem molecular. Para gerenciar os contêineres, a ferramenta Docker foi utilizada, mostrando um gerenciamento de recursos mais eficiente, até mesmo no cenário em que foi alocado mais memória do que realmente disponível fisicamente para as instâncias em execução.

Kominos et al. (2017) fazem uma análise sobre as opções de provisionamento na plataforma OpenStack, que são: máquinas virtuais, contêineres e *bare-metal* (instalação do zero nas máquinas). As alternativas são comparadas em relação à CPU, networking, operações de I/O e acesso à RAM, além de medição de tempo de *boot*. Contêineres Docker container apresentaram o tempo de início mais rápido, e obteve um desempenho muito próximo do *bare-metal* nos outros quesitos, este que por sua vez, teve os melhores resultados. Em geral, VMs obtiveram os piores resultados, principalmente em relação ao desempenho de múltiplas vCPUs.

O artigo de Moltó et al. (2017) descreve um *workflow* para permitir a execução de uma aplicação tanto em ambientes virtualizados, como em ambientes baseados em contêineres, o que foi chamado de Infraestrutura Híbrida de Computação Distribuída (IHCD), utilizando ferramentas open-source do projeto INDIGO-DataCloud. O principal

problema abordado é a falta de interoperabilidade de uma imagem entre hipervisores (VMs) e contêineres. O resultado é uma ferramenta que permite a criação de um *template* com as especificações necessárias da aplicação e a plataforma OpenNebula cria a imagem e a disponibiliza conforme solicitado (Docker ou VM). O tempo total deste processo para uma aplicação de teste é de 20:48 (minutos:segundos) para VM e 7:25 para Docker.

Para finalizar, Li et al. (2017) avaliam as diferenças fundamentais entre Docker e máquinas virtuais, contrariando os outros trabalhos relacionados, porquê apresenta situações onde máquinas virtuais podem apresentar um desempenho melhor que contêineres Docker. Nos benchmarks executados, o Docker atingiu um gargalo de velocidade em operações de disco de textitbyte-a-byte, enquanto os testes com VMs atingiram resultados parecidos com o desempenho nativo para resolver o problema *N-Queens*. Mesmo assim, o trabalho ressalva que as diferenças de *overhead* de desempenho entre os tipos de virtualização ocorrem não somente no modo implementado, como também depende da aplicação em execução.

3.2 Discussão e Lacunas

Na Tabela 1 é feito um resumo dos trabalhos detalhados na subseção 3.1, elencando as principais características dos trabalhos conduzidos. Podemos notar que, apesar do trabalho R. Righi et al. (2016) apresentar um modelo elástico para aplicações PAD, não existem outros estudos que mostrem o comportamento de contêineres sob estas condições. Os demais trabalhos fazem uma análise do desempenho de contêineres, porém sem instanciação dinâmica de contêineres pelo gerenciador, o que daria o aspecto de um sistema elástico.

Os estudos mostrados na Tabela 1 que abordaram a utilização da tecnologia de contêineres, exibem resultados mais favoráveis à estas implementações em comparação ao método tradicionais de virtualização, abordando diversos aspectos inerentes à virtualização de recursos, como operações de I/O, desempenho de memória RAM, CPU e consumo de energia. Todos estes são aspectos fundamentais para uma solução de ambiente PAD eficiente. Embora os trabalhos mostrem resultados aplicando programas de *benchmark* para medir o desempenho da infraestrutura, não foi encon-

trado um estudo que verificasse o desempenho elástico de contêineres para um modelo de provisionamento de recursos de forma dinâmica, como o AutoElastic mostra, utilizando máquinas virtuais.

Tabela 1: Comparação de trabalhos relacionados

Trabalho	Tecnologia	Elasticidade	Base de Comparação	Métricas	Plataforma	Testes
(R. RIGHI et al., 2016)	VM	Sim	Ubuntu 10.04	desempenho, custo e energia	OpenNebula	cálculo de integrais
(CHUNG et al., 2016)	Docker	Não	CentOS 7 3.10	Performance por instancias	OpenStack	MPI com Graph500, Linpack
(XAVIER et al., 2013)	LXC, OpenVZ, VServer	Não	Ubuntu 10.04	diversos tipos de recursos em único nó	Não	Linpack, STREAM, IOZone, NetPIPE
(ZHANG et al., 2016)	Docker	Não	CentOS Linux	desempenho de comunicação	OpenStack	MPI com Graph500, NAS, LAMMPS
(ADUFU et al., 2015)	Docker	Não	Ubuntu 14.04	Desempenho de RAM	OpenStack	*autodock3*
(KOMINOS et al., 2017)	Docker	Não	Ubuntu 14.04	Desempenho de CPU, rede, RAM e disco	OpenStack	ParallelPXZ, Netperf, SysBench
(MOLTó et al., 2017)	Docker	Não	Ubuntu 14.04	Provisionamento	OpenNebula	Tempo médio
(LI et al., 2017)	Docker	Não	Ubuntu 15.10	Desempenho de CPU, disco e RAM	Não	Iperf, STREAM, HardInfo, Bonnie++

Fonte: Elaborado pelo autor

Para o trabalho com aplicações PAD, é muito importante saber todos os aspectos inerentes à infraestrutura que se vai utilizar, como desempenho de CPU, tempo de processamento, energia e custos. Além do hardware, as tecnologias escolhidas devem deixar claro qual seria a mais indicada para o foco da aplicação. Contêineres Docker estão sendo cada vez mais adotados, sendo necessário um estudo para entender as diferenças de desempenho sob estes aspectos, contribuindo para decisões de estudos futuros.

4 BENCHMARK DOCB

Neste capítulo apresentaremos o modelo do Docker Containers Benchmark (DoCB), uma análise de viabilidade de elasticidade em nuvem para contêineres, em comparação ao modelo tradicional, que utiliza máquinas virtuais. Além disto, debateremos em termos gerais as questões de projeto. Na próxima seção 4.2, abordaremos as modificações necessárias no modelo do AutoElastic para suportar este *benchmark*. A modificação no modelo não é o objetivo principal deste projeto, mas se configura com uma contribuição técnica, permitindo uma maior flexibilidade da ferramenta AutoElastic.

4.1 Questões de projeto

O DoCB é uma estratégia de *benchmark*, que segundo Dongarra, Martin e Worlton (1987), é uma técnica de executar um programa (ou conjunto), em uma máquina e comparar o desempenho com outra. A técnica em sua origem era voltada para observação topográfica, onde um objeto estacionário era medido para servir de ponto de referência para observações futuras. Ainda segundo a sua definição, *benchmarking* serve para identificar "quais, de várias opções, irão desempenhar melhor em um dado contexto?". O modelo do DoCB parte de uma observação de um modelo de elasticidade em nuvem utilizando máquinas virtuais, aplicando uma certa carga, para então comparar com a elasticidade com contêineres, dado o cenário de PAD.

Para obter a elasticidade em nuvem, utilizaremos a ferramenta AutoElastic. Nesse caso, iremos estender a ferramenta para permitir a possibilidade de instanciar contêineres além VMs, ficando transparente ao usuário essa opção, minimizando a configuração necessária para utilização das opções de virtualização. A elasticidade se dará no momento em que a aplicação decidir adicionar recursos para otimizar a execução das aplicações, utilizando os critérios definidos. Como as imagens de contêineres possuem um tamanho significativamente menor que o de uma VM, o gerenciador poderá lançar *n* instâncias de contêineres para um mesmo nó, obtendo vantagens ao executar aplicações paralelas.

O gerenciador AutoElastic foi escolhido por permitir virtualização de acordo com a utilização dos recursos computacionais já alocados. A elasticidade aplicada pelo gerenciador será horizontal, ou seja, a elasticidade é aplicada ao alocar e desalocar recursos virtualizados, e não na configuração de hardware. A métrica utilizada pelo gerenciador será a média de CPU das unidades de processamento virtualizadas, e o tempo de entrega destes recursos ao ativar a elasticidade.

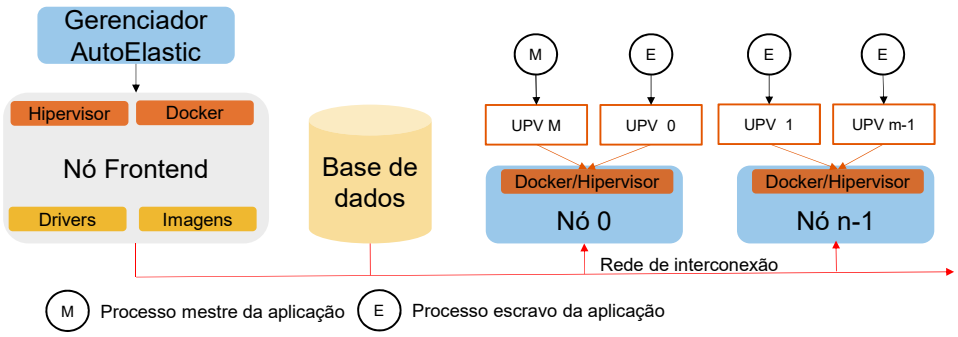
4.2 Extensão da Arquitetura do AutoElastic

Este trabalho parte da implementação do modelo AutoElastic para propor uma modificação em sua arquitetura, acrescentando a possibilidade de utilizar contêineres além de VMs. O AutoElastic é um *middleware*, que permite que o usuário possa obter

elasticidade para suas aplicações, sem depender de alterações no código-fonte. Operando no nível de PaaS, o modelo permite que aplicações iterativas obtenham elasticidade através de operações de alocação e consolidação de instâncias de máquinas virtuais em nós físicos. O *framework* atual é composto por um Gerenciador, um componente independente da infraestrutura, não interferindo no resultado da execução e podendo gerenciar os recursos remotamente, através do acesso de um computador *frontend* da Nuvem de computadores.

A Figura 2 mostra como é a arquitetura do modelo adaptado. Cada nó possui um gerenciador de contêineres Docker, além da implementação corrente com hipervisor. Aqui, utilizaremos o termo Unidade de Processamento Virtualizada (UPV), que pode se referir tanto para uma máquina virtual, quanto um contêiner. O número de UPVs pode partir de 1 por nó, a partir do UPV do processo mestre da aplicação, até vários UPVs por nó, definido pelo usuário do gerenciador (LEE et al., 2011). O gerenciador analisa periodicamente as instâncias de UPVs que estão ativas, e verifica se é necessário um adição ou remoção de recursos, dependendo da configuração de *threshold*. No modelo proposto, o usuário pode definir o número minimo e máximo de UPVs por nó e quantos poderão operar na execução da aplicação. Ainda, a arquitetura garante que a aplicação em execução não tenha o seu desempenho afetado pela adição ou remoção de recursos, pois o gerenciador atua de forma isolada da Nuvem de computadores. A base de dados mostrada na figura 2 é utilizada para armazenar as imagens

Figura 2: Infraestrutura do Modelo. Aqui, *n* representa o número de nós disponíveis, e *m* o número de Unidades de Processamento Virtualizadas (UPV - seja uma máquina virtual ou contêiner).



Fonte: Elaborado pelo autor

utilizadas pelo nó *frontend*, além de permitir um espaço de comunicação entre o gerenciador e as instâncias de UPVs em execução, sendo este espaço acessível apenas

pelos computadores da Nuvem. A utilização de uma área compartilhada, para comunicação entre computadores virtualizados é uma prática comum, quando se falando de nuvens privadas. No contexto de Docker, a base de dados é um banco local *Docker Repository* e a área de dados comum é montada pelos contêineres utilizando a ferramenta de *volumes*, que monta um diretório compartilhado.

4.3 Elasticidade

As aplicações de PAD normalmente lidam com computação intensiva, por esse motivo, iremos nos concentrar apenas na métrica de CPU para tomada de decisão. O Gerenciador captura periodicamente os valores de carga de CPU das UPVs que estão executando os processos. É feita uma média desses valores e estes são armazenados juntamente com coletas operações anteriores para realizar o cálculo de carga geral. O valor resultante do cálculo é avaliado conforme as regras definidas de *threshold* inferior e superior, que são definidos por valores de 0% a 100% respectivamente. O AutoElastic possui em sua implementação algumas opções de algoritmo de avaliação, entre elas o algoritmo que opera segundo o conceito *Aging*. Segundo Tanenbaum (2003), a técnica utiliza uma suavização exponencial simples, aplicando maior peso para a última coleta para definir a carga. Este algoritmo foi escolhido para evitar ruídos nas leituras, que poderiam prejudicar a execução eficiente da aplicação.

4.4 Modelo do benchmarking

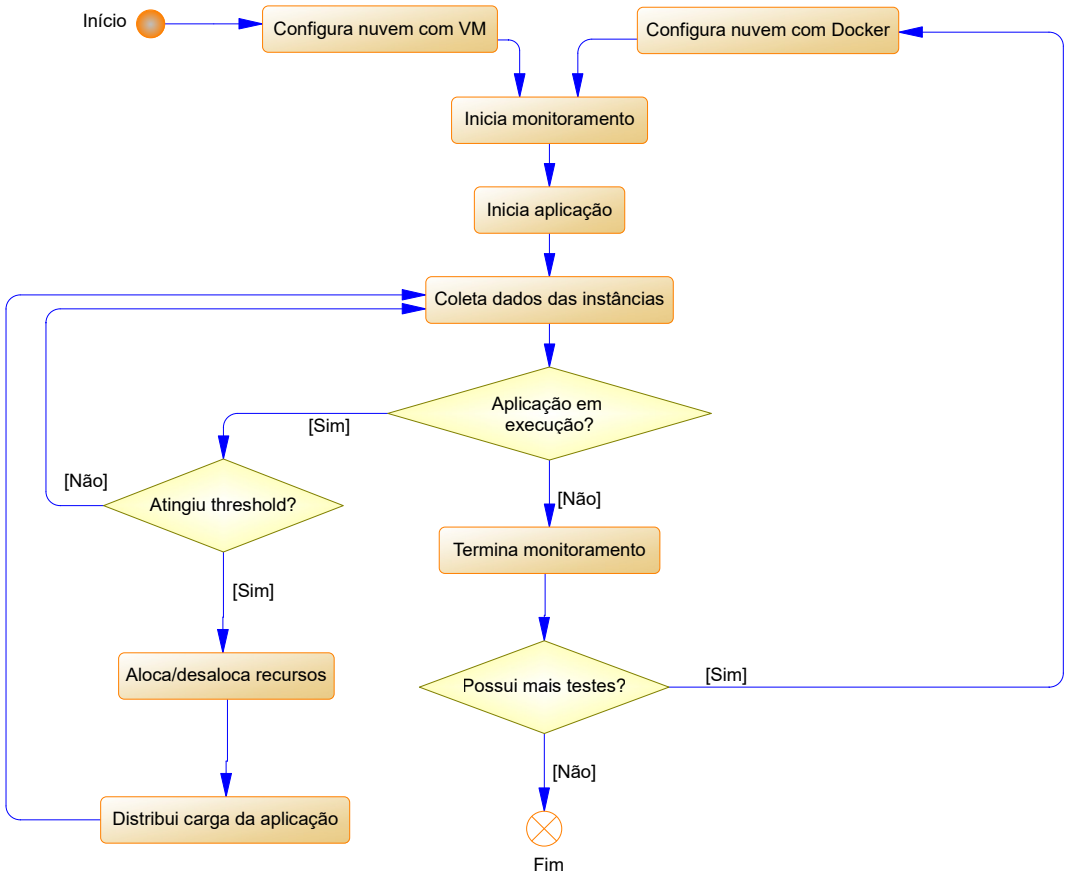
Para poder avaliar o desempenho dos contêineres com operações de elasticidade, será criada uma base de comparação, que se configura por uma execução do processo utilizando máquinas virtuais. Nessa execução, utilizaremos o hipervisor KVM[2] para virtualização. A Figura 3 demonstra o processo de avaliação do *benchkmark* para uma aplicação.

O processo se inicia na configuração do Gerenciador para lidar com máquinas virtuais. A configuração envolve a imagem que irá rodar a aplicação, com especificação de CPU e memória alocados, bem como a rede que esta irá utilizar. Inicia-se então, a

[2] https://www.linux-kvm.org

aplicação e o monitoramento da Nuvem, que identifica os recursos ativos e os processos em execução. A coleta de dados é feita no próximo passo, que a cada intervalo de 15 segundos, avalia a carga de CPU das instâncias alocadas, utilizando estes dados para a decisão. Os *thresholds* superior e inferior são avaliados para decidir alocar ou desalocar recursos para a aplicação. No caso de alocar, um número *x* de UPVs (VM ou contêiner) são instanciados para uma nova máquina carregada e a carga da aplicação é distribuída entre todos os recursos disponíveis.

Figura 3: Processo de coleta de dados do benchmarking, avaliando a execução de uma aplicação com máquinas virtuais e contêineres



Fonte: Elaborado pelo autor

Na coleta de dados, também é verificado se a aplicação já terminou o processamento, finalizando esta etapa do processo e iniciando a etapa de execuções com contêineres, que pode vir a ser de *n* iterações, dependendo da intenção do usuário. A ferramenta ONEDock[3] é utilizada para configurar a Nuvem para suportar contêineres Docker. O processo acaba quando se esgota as iterações da bateria de testes.

[3] https://github.com/indigo-dc/onedock

Uma imagem de contêiner Docker é criada a partir de um arquivo chamado *Dockerfile*, que contém instruções de configuração ao criar a imagem. Esta imagem pode ser extremamente pequena, se comparada à uma imagem virtualizada tradicional, ocupando poucos megabytes. Outra peculiaridade dos contêineres é que estes não possuem configuração de limitação de recursos por padrão, ou seja, um contêiner irá utilizar tanta CPU e memória da máquina, quanto for disponibilizado pelo agendador do *kernel*. Esta característica nos permite criar um *template* diferente para os contêineres, sem limitação de recursos, diferentemente de VMs, que precisam ter definido o quanto de memória e CPU da máquina que irão utilizar.

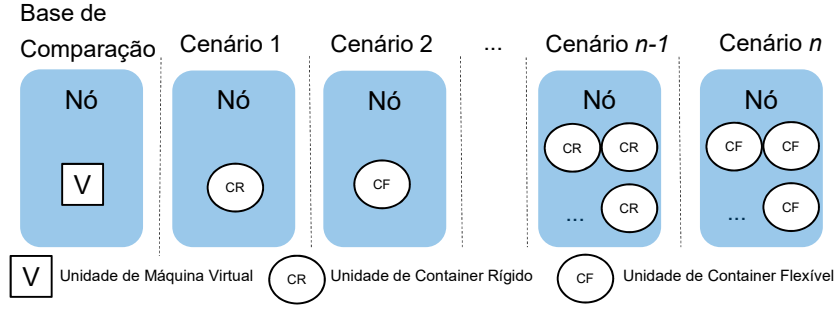
Para cada execução da aplicação com contêineres, iremos configurar sempre um par de *templates*:

- Rígido: Ao iniciar um contêiner, iremos especificar quantos ciclos de máquina da CPU (subconjunto de MHz da capacidade teórica do processador em questão) o contêiner poderá utilizar, mesmo comportamento que uma máquina virtual.

- Flexível: Iremos iniciar o contêiner em um processador, sem especificar limite de recursos. Os ciclos de máquina da CPU serão obtidos sob demanda, sendo concorrente com outros contêineres e processos, sem isolamento.

Sendo assim, podemos explorar vários cenários com contêineres, após termos o cenário base com máquinas virtuais. O modelo do benchmark especifica a abordagem mostrada na Figura 4. Nela podemos identificar que diferentes números de instâncias de contêineres podem ser exploradas para popular os nós nos cenários, sendo que sempre será executado uma vez com formato Rígido e outro com o Flexível.

Figura 4: Modelo de cenários para execução do benchmarking, onde n representa o limite de cenários para contêineres



Fonte: Elaborado pelo autor

5 METODOLOGIA DE AVALIAÇÃO

Nesta seção, serão apresentadas as modificações necessárias no AutoElastic para suportar contêineres. Após isto, será mostrado como o modelo do *benchmark* foi implementado. As especificações da implementações englobam a aplicação utilizada, os cenários que foram escolhidos e detalhes técnicos da infraestrutura utilizada.

5.1 Implementação

Para a execução dos experimentos em nuvem, utilizaremos o protótipo desenvolvido por R. Righi et al. (2016), que consiste de uma aplicação em Java para gerenciamento de elasticidade, o AutoElastic. O modelo será estendido, passando a considerar a utilização de contêineres. O protótipo atual consiste de uma ferramenta que gerencia uma nuvem OpenNebula, sendo o controle do ambiente realizado através de uma API em Java que o *middleware* OpenNebula oferece, como por exemplo, a criação de máquinas virtuais dentro da Nuvem.

O ONEDock é um conjunto de extensões para o OpenNebula para utilização de contêineres Docker como entidades de primeira classe, assim como se fossem máquinas virtuais leves. Para tal, Docker é configurado para atuar como um hipervisor, comportando-se então, como o KVM faz no contexto de OpenNebula (ALFONSO LAGUNA, 2015). Este conjunto de ferramentas é open-source, o que possibilitou a codificação de características não suportadas pela distribuição atual da ferramenta, como por exemplo a especificação de CPU e memória limitada para o contêiner.

5.2 Cenários de avaliação

Para a execução dos cenários de teste, utilizaremos a aplicação desenvolvida por R. Righi et al. (2016), o programa consiste de um algoritmo para rodar cálculos de integrais em uma rede privada, seguindo o modelo mestre-escravo, sob diferentes tipos de comportamento de carga de computação. No nosso caso, utilizaremos as cargas ascendente e descendente. A aplicação realiza cálculos numéricos e distribui porções de trabalho para cada nó, interligados por MPI (*Message Passing Interface*).

O processo mestre fica ciente de novos nós disponíveis utilizando um sistema de comunicação de troca de arquivos de texto, disponibilizados pela aplicação AutoElastic durante as operações de elasticidade. O objetivo da aplicação é puramente demonstrar as operações de elasticidade do gerenciador, considerando as variações de carga e o impacto do assincronismo no desempenho de aplicações.

5.2.1 Definição dos testes

O processo inicia com uma execução da aplicação utilizando máquinas virtuais. Para tal, dois templates de imagens foram configurados com o sistema operacional Ubuntu 10.10, sendo o mestre provido com 1 unidade de CPU e 1GB de RAM, além do template escravo com 1 unidade de CPU e 2GB de RAM, desta forma, cada vez que for solicitado por um recurso adicional de computação, uma unidade escravo é carregada para um dos nós de computação, ocupando 1 dos cores de CPU e metade da memória RAM disponível.

A execução da aplicação paralela com configuração de 2 VMs por nó (1 VM por core) é considerada otimizada, já que cada core se responsabiliza por um processo (R. RIGHI et al., 2016). Porém, não sabemos se a mesma regra se aplica ao cenário de contêineres, por serem processos mais leves que uma máquina virtual, é possível que um número maior de contêineres por nó tenha um desempenho melhor. Tendo em vista isto, criamos os cenários de acordo com a Tabela 2.

No cenário 1 da Tabela, temos o que seria uma configuração considerada equivalente ao definido para o cenário de VMs, ou seja, 2 contêineres por operação, sendo cada um com 1 core da CPU e 2GB de memória RAM. O cenário 2 também inicia o processo com 2 contêineres na máquina, porém sem especificação de recursos, deixando a máquina alocar processamento para os processos, conforme necessário. Foram criados outros 2 cenários para avaliar a capacidade de paralelização dos contêineres, ambos com suas variações de modelo Rígido e Flexível em relação à limitação de recursos.

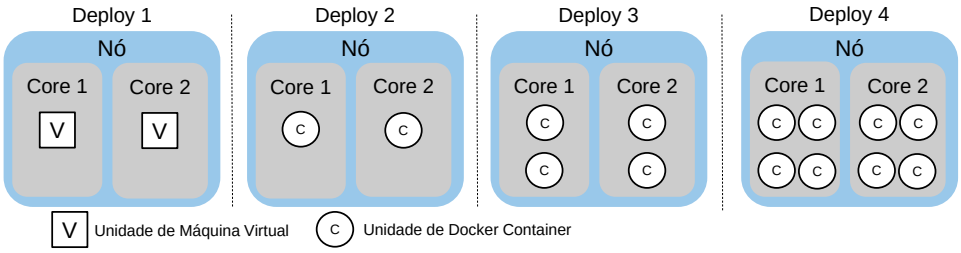
Na Figura 5 podemos identificar como o gerenciador trata cada uma das variações de cenários de *deploy* (entrega da UPV pronta para o nó). O *deploy* 1 é a primeira etapa do processo, entregando duas máquinas virtuais para o recurso alocado. Poste-

Tabela 2: Cenários de contêineres

Cenário	Modelo	CPU alocada	RAM alocada (MB)	contêineres por operação
1	Rígido	1	2048	2
2	Flexível	Não definido	Não definido	2
3	Rígido	0.5	1024	4
4	Flexível	Não definido	Não definido	4
5	Rígido	0.25	512	8
6	Flexível	Não definido	Não definido	8

riormente, temos o *deploy* 2, no qual a mesma quantidade de contêineres é entregue para uma nova máquina alocada, este processo ocorre tanto para o modo Rígido quanto para o modo Flexível de limitação de CPU, sendo que o Rígido é o equivalente à VMs. Na sequência, o *deploy* 3 é executado, entregando 4 contêineres para uma máquina alocada, novamente com os dois modos (Rígido e Flexível). Finalmente, um último teste é executado entregando 8 contêineres por operação de elasticidade.

Figura 5: Cenários de *deploy* de instâncias de máquinas virtuais e contêineres para uma mesma máquina



Fonte: (DOCKER, 2016)

5.2.2 Infraestrutura

Como infraestrutura de Nuvem, utilizaremos o ambiente disponibilizado pela Universidade do Vale do Rio dos Sinos, localizado no laboratório C01 413 do Programa de Pós-Graduação em Computação Aplicada (PIPCA). O laboratório conta com 18 computadores, interconectados através de uma rede 100Mbps, sendo a configuração de cada um deles uma memória de 4 GB, além de processadores de dois núcleos de 2.9 GHz. Porém, para fins de teste de cenários específicos deste trabalho, 5 máquinas do laboratório foram configuradas para serem utilizadas. A plataforma de nuvem OpenNebula será instalada nesta rede de computadores para a execução do protó-

tipo, sendo que uma das máquinas será utilizada como nó *Front-End*. A versão 1.1 do ONEDock tem suporte mínimo para a distribuição 4.14 do OpenNebula, portanto esta também teve que ser instalado. Além disto,a versão 1.9 do Docker foi escolhida por ser a versão para qual o ONEDock 1.1 possui suporte até o presente momento.

5.3 Métricas e Parâmetros

De forma a contemplar os trabalhos relacionados, os valores 80% e 40% foram escolhidos para os *thresholds* superior e inferior. Como métricas de desempenho, foram escolhidos o tempo total de execução, a energia gasta e o custo de processamento. A energia é avaliada de acordo com o número de máquinas em cada observação, podemos ver na Equação 1 , onde i é número de máquinas físicas no tempo $T(i)$. O custo é mostrado na Equação 2, que é uma função de adaptação do custo de computação paralela para ambientes elásticos (WILKINSON; ALLEN, 2004). O objetivo desta métrica é identificar se uma determinada configuração é viável em termos de custo, mas se pode não ser aceitável em questão de tempo total de execução.

$$Energia = \sum_{i=1}^{n} (i \times T(i)) \tag{1}$$

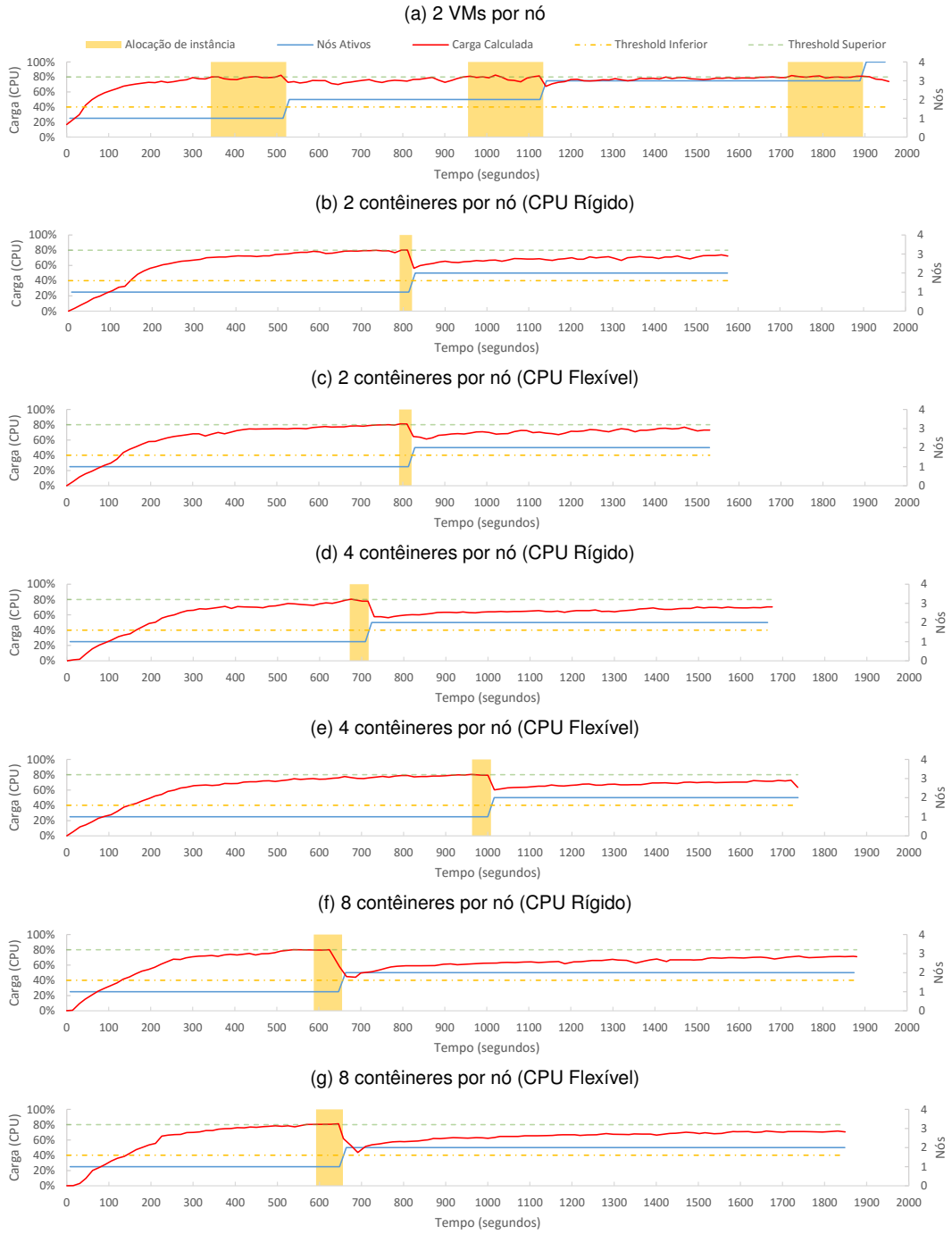
$$Custo = Tempo \times Energia \tag{2}$$

6 RESULTADOS

Nesta seção, serão apresentados os resultados dos testes executados, conforme descrito anteriormente. Inicialmente, é possível observar que a Figura 6 mostra os resultados obtidos durante os testes com carga ascendente. Como pode ser observado em (a), quando a operação de elasticidade é disparada para a infraestrutura com máquinas virtuais - no threshold de 80% - ocorre um tempo de espera de 180 segundos até que as 2 novas VMs estejam prontas para uso da aplicação. Este mesmo comportamento ocorre para as outras duas operações subsequentes de elasticidade no mesmo teste.

Ao utilizar 2 contêineres por operação de elasticidade, com o modelo Rígido de CPU em (b), é possível observar que a operação de elasticidade (instanciar 2 novos

Figura 6: Tendência de tempo de execução de aplicação **ascendente**, variando o número de instâncias virtualizadas por nó alocado



Fonte: Elaborado pelo autor.

contêineres) demorou 30 segundos, um número muito menor em comparação com as máquinas virtuais. Além disto, é importante observar que: 2 operações de elasticidade não chegam a ocorrer - a aplicação rodou em apenas 2 nós, ao invés de 4 - e que mesmo assim, o tempo total de execução foi de 380 segundos a menos. No teste com

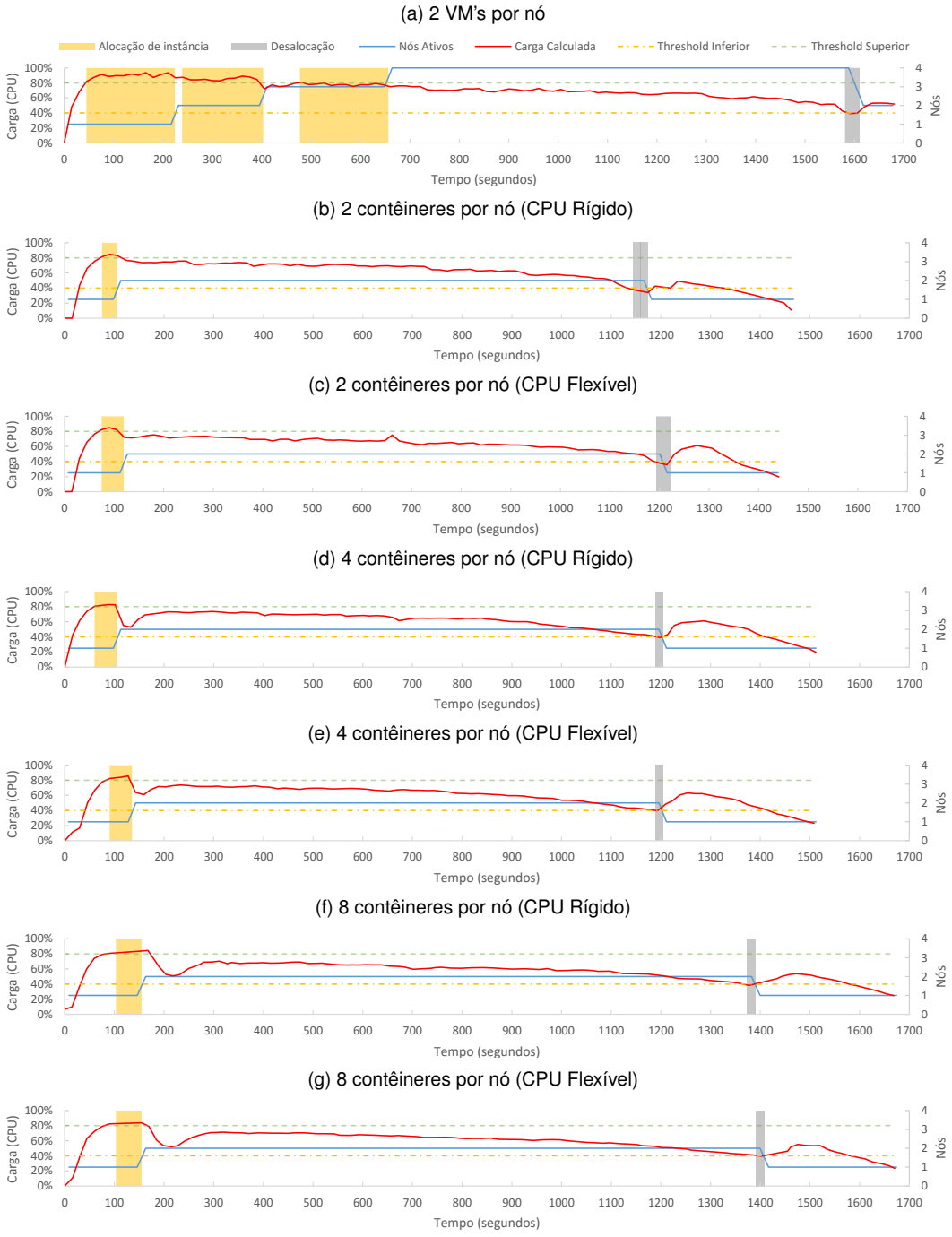
2 contêineres utilizando o modelo Flexível de CPU (c), não foi observado diferença nas operações de elasticidade, mas como esperado, ocorreu um tempo menor de execução da aplicação, em torno de 45 segundos a menos. Quando se aumentou o número de contêineres por operação (d)(e)(f)(g), notou-se um aumento linear, tanto na duração da operação de elasticidade, como no tempo total da aplicação.

Em todas as operações com contêineres, percebeu-se uma diferença significativa em relação ao teste com VMs. O que corrobora com a questão do *overhead* na utilização de máquinas virtuais, conforme os argumentos dos trabalhos relacionados. Porém, como no nosso caso, esse *overhead* pode ser a diferença entre ser necessário alocar um recurso a mais ou não. É possível observar essa sobrecarga logo no começo das execuções: para o teste com máquinas virtuais (a), a carga chega aos 40% logo nos 45 segundos de execução, já para todas as outras execuções com contêineres, isto não ocorre antes dos 130 segundos de execução.

Na Figura 7, é possível observar as técnicas de virtualização trabalhando com carga descendente. Neste cenário, o mesmo comportamento de carga se repete, exigindo mais processamento com máquinas virtuais. No teste com operações de 2 VMs (a), temos 3 operações de elasticidade, praticamente em sequência, ficando com 4 nós alocados. Posteriormente, a carga fica abaixo do *threshold* inferior (40%) e a desalocação de 2 máquinas virtuais é acionada. O tempo de desalocar os recursos tende a ser o mesmo utilizando contêineres (b), demorando em média de 15 a 30 segundos, o que se refere ao tempo médio de 1 a 2 intervalos de verificação pelo gerenciador.

É possível notar que, assim como nos testes com carga ascendente, a execução com 2 contêineres, utilizando CPU Flexível (c), é a que apresentou um menor tempo total de execução, mesmo com apenas 1 adição de nó, gerando uma diferença de aproximadamente 300 segundos. Esta execução, assim como as outras com contêineres, finalizou com carga de CPU abaixo dos 40%, porém a operação de desalocação não ocorre, pois foi configurado no gerenciador para existir no mínimo um nó executando a aplicação, até ela terminar o processamento.

Figura 7: Tendência de tempo de execução de aplicação **descendente**, variando o número de instâncias virtualizadas por nó alocado, e atingindo *thresholds* superior e inferior



6.1 Análise

A Figura 8 mostra um comparativo das métricas coletadas de cada estratégia de virtualização, durante os testes executados. Conforme já destacado nos resultados anteriores, as métricas confirmam o desempenho superior para a virtualização com

contêineres para as aplicações de carga ascendente e descendente. Como pode ser observado na Figura 8, tanto o tempo total de execução, quando a energia calculada, são menores para os contêineres, o que reflete num valor de custo calculado bastante discrepante.

No cálculo de custo para a aplicação ascendente (c), a execução com 2 contêineres, utilizando CPU Flexível, apresentou uma diferença de 59% em relação ao teste com 2 máquinas virtuais. Se formos comparar os dois modelos de CPU (Rígido e Flexível), para uma mesma execução de contêineres, poderemos notar um valor de 6,5% menor para o teste de 2 contêineres com CPU Flexível, em comparação ao teste com 2 contêineres com CPU Rígido. No cálculo de custo para a aplicação descendente (f), o teste com 2 contêineres e CPU Flexível mostrou um ganho de 60% em comparação com execução com 2 máquinas virtuais. A grande diferença de custo, no casos desses testes, se dá principalmente pelo maior consumo de energia pelas máquinas virtuais. Dado fato de ter sido necessário dois nós a mais para cada tipo de aplicação, o que influenciou num custo calculado muito maior para esta estratégia de virtualização.

Figura 8: Desempenho para aplicação ascendente (cont. R. abrevia para contêineres rígidos e cont. F. para contêineres flexíveis para limite de CPU utilizada)

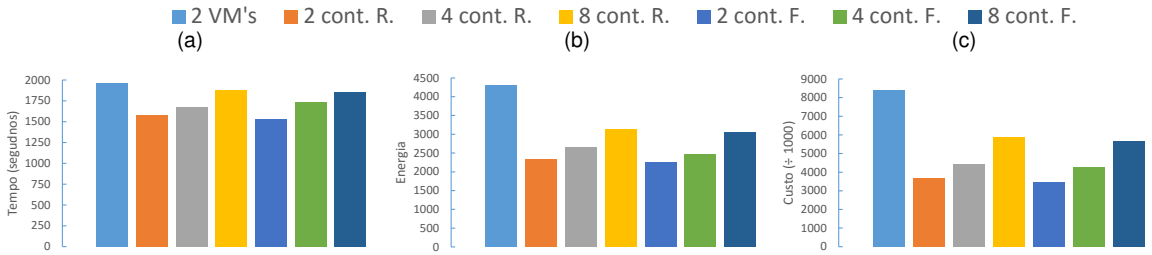
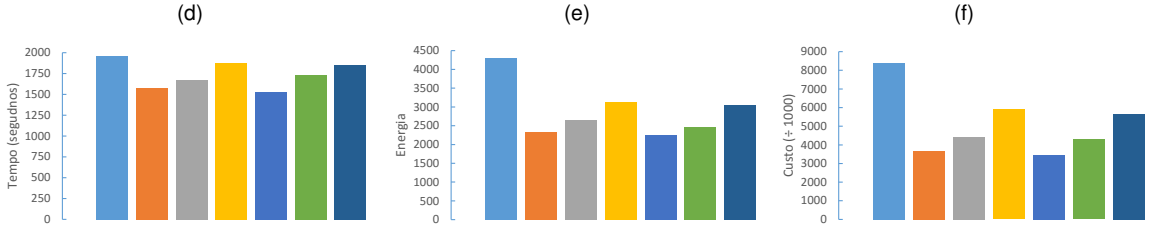


Figura 8: Desempenho para aplicação descendente (cont. R. abrevia para contêineres rígidos e cont. F. para contêineres flexíveis para limite de CPU utilizada)



7 CONCLUSÃO

Neste trabalho, inicialmente foi realizado uma pesquisa do estado-da-arte de virtualização de recursos para ambientes PAD e *benchmarks* em cima da utilização de contêineres e máquinas virtuais. A partir da pesquisa, foi identificado uma lacuna na literatura quanto à diferença que pode ter ao utilizar uma tecnologia ou outra em ambientes de Nuvem com elasticidade. Com o objetivo de preencher essa lacuna, este artigo propôs o DoCB, um *benchmarking* sobre as estratégias de virtualização e o impacto na Nuvem com elasticidade. O modelo propõe uma série de execuções de uma aplicação em Nuvem com elasticidade, partindo de uma abordagem com máquinas virtuais, para formar a base de comparação, seguindo de execuções variando o número de contêineres por nó, bem como variando também o limite de CPU por contêiner (Rígido e Flexível), um recurso que não existe na virtualização com hipervisores. O diferencial deste *benchmarking* em relação aos outros trabalhos, é a exploração da elasticidade e alocação dinâmica de novos recursos. Outra contribuição foi a análise de desempenho de contêineres variado o número de instâncias por nó e considerando vantagens específicas de contêineres, como o limite de CPU Flexível.

A fim de executar o *benchmarking* proposto, foi feita uma outra contribuição técnica, que é a adaptação do gerenciador de Nuvem (AutoElastic), para suportar a utilização de contêineres. O gerenciador atuou com os vários cenários de virtualização para uma aplicação com dois tipos de carga: ascendente e descendente. Os resultados foram avaliados segundo as métricas de tempo total de execução, energia e custo (tempo x energia). Nos testes com carga ascendente, a utilização de 2 contêineres por nó com CPU Flexível, apresentou um tempo de aproximadamente 20% menor que o teste com 2 máquinas virtuais. Ainda, por causa da energia gasta com nós adicionais, a mesma execução com contêineres apresentou um custo de aproximadamente 60% menor que a execução com máquinas virtuais.

Como sugestão de trabalhos futuros, indica-se a análise de aplicações variando *thresholds* e explorar outros aspectos de contêineres Docker. Outra possibilidade é a implementação em um ambiente produtivo e verificar o comportamento do gerenciador com as técnicas de virtualização utilizando cargas reais.

THE TITLE IN ENGLISH

Abstract: Este documento apresenta orientações para uso da classe LaTeX de formatação de artigos para a UNISINOS. Ao mesmo tempo, ele serve como exemplo de uso da classe, demonstrando os principais comandos a serem utilizados, e outras orientações mais gerais de uso do LaTeX. Adicionalmente, procuramos incluir no documento algumas orientações sobre a escrita da monografia em si, reunindo dicas e recomendações que contribuem para aumentar a qualidade técnica dos trabalhos acadêmicos. O Resumo deve conter de 150 a 250 palavras e apresentar o objetivo, o método, os resultados e as conclusões do artigo. Deve ser composto por frases concisas e afirmativas. Recomenda-se o uso de parágrafo único. Deve-se usar o verbo na voz ativa e na terceira pessoa do singular.

Keywords: UNISINOS. ABNT. Formatação de documentos. LaTeX.

REFERÊNCIAS

ADUFU, T. et al. Is container-based technology a winner for high performance scientific applications? **IEICE - Dept. of Computer Science, Sookmyung Women's University, Seoul, Korea**, [S.l.], 2015.

ALFONSO LAGUNA, C. de. **Opennebula docker driver and datastore**. Disponível em https://opennebula.org/opennebula-docker-driver-and-datastore/. Acesso em dezembro de 2016.

BERNSTEIN, D. Containers and cloud: from lxc to docker to kubernetes. **IEEE Cloud Computing**, [S.l.], v. 1, n. 3, p. 81–84, Sept 2014.

CHUNG, M. T. et al. Using docker in high performance computing applications. In: IEEE SIXTH INTERNATIONAL CONFERENCE ON COMMUNICATIONS AND ELECTRONICS (ICCE), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p. 52–57.

COUTINHO, E. et al. Elasticidade em computação na nuvem: uma abordagem sistemática. **XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2013)-Minicursos**, [S.l.], 2013.

DOCKER. **The definitive guide to docker**. [S.l.]: www.docker.com, 2016.

DONGARRA, J.; MARTIN, J. L.; WORLTON, J. Computer benchmarking: paths and pitfalls: the most popular way of rating computer performance can confuse as well as inform; avoid misunderstanding by asking just what the benchmark is measuring. **IEEE Spectrum**, [S.l.], v. 24, n. 7, p. 38–43, July 1987.

DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support paas. **IEEE International Conference on Cloud Engineering**, [S.l.], 2014.

DUNBAR, J. **Nas parallel benchmarks**. Disponível em https://www.nas.nasa.gov/publications/npb.html. Acesso em outubro de 2016.

GRABER, S. **Linuxcontainers.org infrastructure for container projects**. Disponível em https://linuxcontainers.org/. Acesso em setembro de 2016.

HAN, R. et al. Lightweight resource scaling for cloud applications. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING (CCGRID 2012), 2012., 2012. **Anais...** [S.l.: s.n.], 2012. p. 644–651.

KOMINOS, C. et al. Bare-metal, virtual machines and containers in openstack. In: CONFERENCE ON INNOVATIONS IN CLOUDS, INTERNET AND NETWORKS (ICIN), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 36–43.

LEE, Y. et al. Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2011., 2011. **Anais...** [S.l.: s.n.], 2011. p. 129–140.

LI, Z. et al. Performance overhead comparison between hypervisor and container based virtualization. In: IEEE 31ST INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS (AINA), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 955–962.

MAUCH, V.; KUNZE, M.; HILLENBRAND, M. High performance cloud computing. **Future Generation Computer Systems**, [S.l.], v. 29, n. 6, p. 1408 – 1416, 2013.

MELL, P.; GRANCE, T. The nist definition of cloud computing: recommendations of the national institute of standards and technology (2011). **URL http://csrc. nist. gov/publications/nistpubs/800-145/ ...**, [S.l.], 2012.

MOLTó, G. et al. Coherent application delivery on hybrid distributed computing infrastructures of virtual machines and docker containers. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 486–490.

NICKOLOFF, J. **Docker in action**. [S.l.]: Manning, 2016.

PAHL, C. Containerization and the paas cloud. **IEEE Cloud Computing**, [S.l.], v. 2, n. 3, p. 24–31, May 2015.

R. RIGHI, R. d. et al. Autoelastic: automatic resource elasticity for high performance applications in the cloud. **IEEE Transactions on Cloud Computing**, [S.l.], v. 4, n. 1, p. 6–19, Jan 2016.

ROUMELIOTIS, B. W. **Cloud architecture patterns**. [S.l.]: O'Reilly Media, Inc, 2012.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. **Computação em nuvem**: conceitos, tecnologias, aplicações e desafios. 2010. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Ceará (UFC), 2010.

SULEIMAN, B. et al. Tradeoff analysis of elasticity approaches for cloud-based business applications. **Web Information Systems Engineering-WISE**, [S.l.], p. 468–482, 2012.

TANENBAUM, A. S. **Computer networks**. 4. ed. Upper Saddle River: Prentice Hall PTR, 2003. 892 p.

TAURION, C. **O que é elasticidade em cloud computing?** Disponível em https://goo.gl/k1L4AC. Acesso em outubro de 2016.

TOSATTO, A.; RUIU, P.; ATTANASIO, A. Container-based orchestration in cloud: state of the art and challenges. In: NINTH INTERNATIONAL CONFERENCE ON COMPLEX, INTELLIGENT, AND SOFTWARE INTENSIVE SYSTEMS, 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p. 70–75.

WILKINSON, B.; ALLEN, M. **Parallel programming**: techniques and applications using networked workstations and parallel computers. [S.l.]: Pearson/Prentice Hall, 2004.

XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. **21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**, [S.l.], 2013.

ZHANG, J. et al. Performance characterization of hypervisor- and container-based virtualization for hpc on sr-iov enabled infiniband clusters. **IEEE International Parallel and Distributed Processing Symposium Workshops**, [S.l.], 2016.