Container Linux, uma Implementação Web Amigável

Marco Otávio, Brivaldo Junior

[1]Faculdade de Computação (Facom) – Universidade Federal Mato Grosso do Sul (UFMS)
79.070-900 – Campo Grande – MS – Brazil

otavio.marco91@gmail.com, brivaldo@facom.ufms.br

Abstract. *Virtualized systems have flooded the market and universities in the last* 5 *years. Technologies such as the mainframe, previously abandoned or restricted to processing environments of large size, now return with mechanisms capable of executing dozens of systems in a single computer in a dynamic and elastic way. However, while virtualization has several advantages over the use of physical hardware, there is still a high computational cost because each virtualized system has its own kernel, process manager, allocation of RAM, disk, etc. Containers have emerged as a light alternative to pure virtualization because they use the same host system kernel in caged environments, delivering more faster abstraction to start, run and provide load balancing. A host, using containers, jumps from tens to hundreds or even thousands instances on the same hardware. This work presents the implementation and modification of LXDUI, a friendly way to guarantee access to containers for users.*

Resumo. *Sistemas virtualizados inundaram o mercado e as universidades nos últimos* 5 *anos. Tecnologias como mainframe, antes abandonadas ou restritas aos ambientes de processamento de grande porte, agora retornam com mecanismos capazes de executar dezenas de sistemas em um único computador de forma dinâmica e elástica. Contudo, embora a virtualização tenha várias vantagens sobre o uso de hardware físico, ainda existe um custo computacional alto pois cada sistema virtualizado tem seu próprio kernel, gerenciador de processos, alocação de memória RAM, disco, etc. Containers surgiram como uma alternativa leve a virtualiação pura, pois usam o mesmo kernel do sistema hospedeiro em ambientes enjaulados, fornecendo uma abstração mais rápida para iniciar, executar e fornecer balanceamento de carga. Um host, usando containers, salta de dezenas para centenas ou até milhares de instâncias em um mesmo hardware. Este trabalho apresenta a implementação e modificação do LXDUI, uma forma amigável de garantir acesso aos containers para usuários.*

1. Introdução

Ao longo dos anos, a crescente demanda por sistemas de informação, tanto comerciais como acadêmicos ou de pesquisa, fizeram com que as empresas repensassem como esses serviços seriam distribuídos para os mais variados usuários. O alto custo atrelado a manutenção de *hardware*, reduz a elasticidade para suprir a demanda por mais serviços ou agilidade na resposta a requisições na Internet. A virtualização, tecnologia herdada da época dos *mainframes*, retorna com objetivo de utilizar *hardware commodity* para entregar com menos infraestrutura.

Neste cenário pode-se elencar tecnologias como VMware [VMware 2018], HyperV [Microsoft 2018], XEN [Mattos 2008], dentre outros. O problema maior com a virtualização é que ela introduz gargálos que não puderam ser totalmente sanados, como a emulação da BIOS/EFI do sistema operacional, sobrecarga no gerenciamento de processos (virtualizador realiza agendamento de processos das máquinas virtuais que, por sua vez, agendam os serviços dentro da cada uma).

Dessa forma, embora prático, a virtualização ainda apresentava desafios que tornavam a tecnologia restrita e dependente de emulações de dispositivos físicos e gerenciamento avançado de memória, disco, cotas de uso, etc. Os *containers* vieram para resolver, em ambientes Unix-*like*, este problema. O Linux, em especial, implementou no kernel a capacidade de duplicar tabelas de endereçamento de memória, acesso a disco e rede, da tal forma que o tornou capaz de criar jaulas completas capazes de entregar mini sistemas virtualizados leves.

O Docker [Vaughan-Nichols 2014], sistema de containers usado nos servidores do Google, foi criado para executar apenas um serviço enjaulado e garantir que os desenvolvedores pudessem definir quais versões de aplicações e bibliotecas uma página web usaria, além da elasticidade de serviços usando *swarm* ou *kubernetes* [Foundation], o LXD [Banerjee 2014] entrega um sistema Linux minimalista com `init` (capacidade de executar múltiplos serviços) com uma interface similar a uma máquina virtual.

As principais diferenças de uma máquina virtual para os containers são o tempo de inicialização (nos containers pode chegar a 96% mais rápido), simplificação (pois usa os *drivers* e *hardware* do sistema hospedeiro) e capacidade elástica (uma vez que o tempo de inicialização é curto, migrar, parar ou reiniciar é mais rápido que máquians virtuais convencionais totalmente emuladas).

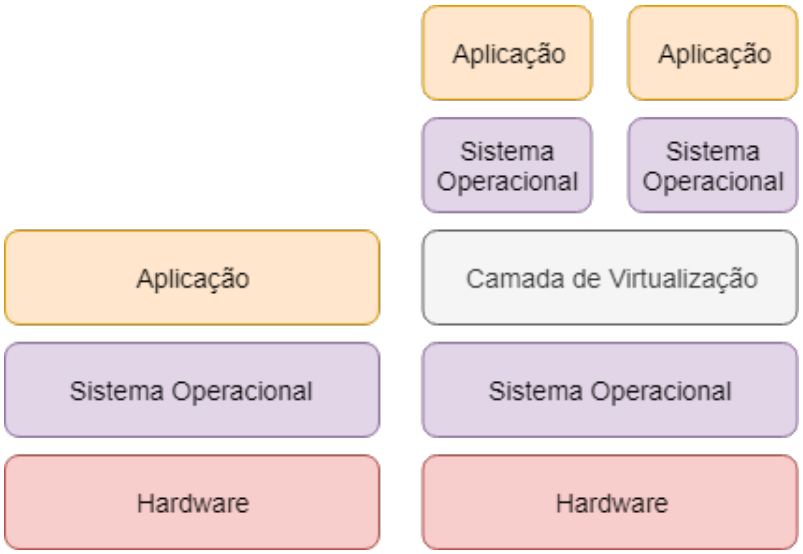

Figura 1. Arquitetura tradicional vs. Arquitetura virtual

Comparando com a virtualização, existem problemas em sua adoção: alta demora de emulação, pois o *kernel* do sistema é executado sobre o *hipervisor* (Hypervisor), sendo este um *software* que é executado entre o sistemas operacional e o *hardware* e é responsável por abstrair a troca de informações entre o sistema hospeiro e o hospedado. So-

brecarga na inicialização do sistema virtual, uso de *hardware* virtual, o sistema virtual usa *drivers* que emulam o *hardware* real, que enviam as interrupções para o *hipervisor*. Atraso de rede imposto por um comutador virtual, como o OpenVSwich [Pfaff et al. 2015] e alto uso dos recursos de máquina o que prejudica diretamente o desempenho de processamento e memória.

Por outro lado, a virtualização possui qualidades. Capacidade de instalar virtualmente qualquer sistema operacional. Isso é possível porque o *hipervisor* fornece uma API (*Application Programming Interface*) que permite ao sistema virtual ”entender”que é uma máquina real. Por isso é possível instalar sistemas Windows, Linux ou MacOS em máquinas virtuais. No entanto, uma camada é criada entre o sistema operacional virtual e o hospedeiro (Figura 1).

Diversos projetos mais conhecidos do público em geral como Docker e VMWare, já fornecem interfaces amigáveis para controle de informações e recursos desses containers. Entretanto, alguns outros projetos como é o caso do LXD, não fornecem nativamente ou fornecem poucos recursos para execução desse gerenciamento. Ou seja, não existe uma interface amigável ao usuário para controlar seus containers.

Esse projeto propõe uma modificação em uma interface web para o LXD [Banerjee 2014] nomeado de LXDUI [AdaptiveScale 2018]. Essa ferramenta permite que um administrador controle todos os aspectos do ambiente, dos mais básicos (imagens que podem ser usadas, usuários que terão acesso, etc) aos mais avançados (limitação de recursos de processador e memória ou tempo de vida do container).

Este trabalho está assim organizado: Na Seção 2 será apresentada a fundamentação teórica a respeito de Linux Containers. Na Seção 3 serão apresentados trabalhos relacionados. Na Seção 4 as modificações realizadas no projeto base. E na Seção 5 a conclusão e trabalhos futuros.

2. LXC - Linux Containers

O Linux Containers (LXC) é uma tecnologia de container a nível de sistema operacional, em específico sob o kernel Linux. Além de permitir a criação de ambientes totalmente virtuais (VE - *Virtual Environment*) é possível executá-los isoladamente em contêineres e demais aplicativos podem usá-los em forma de *sandbox* ou emular uma máquina totalmente nova [Baukes 2017].

O kernel Linux [Oliveira 2017] é o *software*, responsável por toda a troca de informação entre entre uma aplicação e o *hardware*, tem, como uma de suas funcionalidades, o *cgroups*. Essa função, combinada com o *namespaces*, que é um conjunto de símbolos para organização de objetos no sistema operacional, permite a criação de grupos de controle de usuários. Deste modo, o particionamento de recursos - disco, memória, processador etc - se torna extremamente eficiente, o que contribui para que o LXC se torne uma alternativa viável para instituições que necessitam de ter várias máquinas ativas.

O excelente gerenciamento de recurso provido pelo kernel Linux, fornece ao LXC uma vantagem quando comparada as máquinas virtuais convencionais. Estas, quando criadas, tem suas informações, sistema operacional, drivers etc empacotados, o que permite a portabilização do ambiente conforme conveniência do administardor do sistema, além do baixo tempo de *boot* e pouca sobrecarga de memória.

3. Trabalhos relacionados

3.1. Docker

O Docker é uma tecnologia de containers desenvolvida afim de facililtar a criação, manutenção e gerenciamento de containers. Esse *software* fornece uma camada adicional de abstração, permitindo o empacotamento total da aplicação virtualizada, ou seja, todos os containers virtualizados são isolados (Figura 2). Para execução desse gerenciameno o Docker utiliza, além da tecnlogia *cgroups* e *namespace* o LXC [Vaughan-Nichols 2014].

Por ser totalmente isolado um container Docker é altamente portável, ou seja, um ambiente inteiro pode ser alocado em outro host em caso de mudança ou manutenção. Isso permite, também, que o tempo de *deploy* do ambiente ou da aplicação seja reduzido drasticamente visto que não há necessidade de ajustes para execução. Como utiliza LXC no *backend*, é possível estabelecer limites de I/O, processador, memória etc.

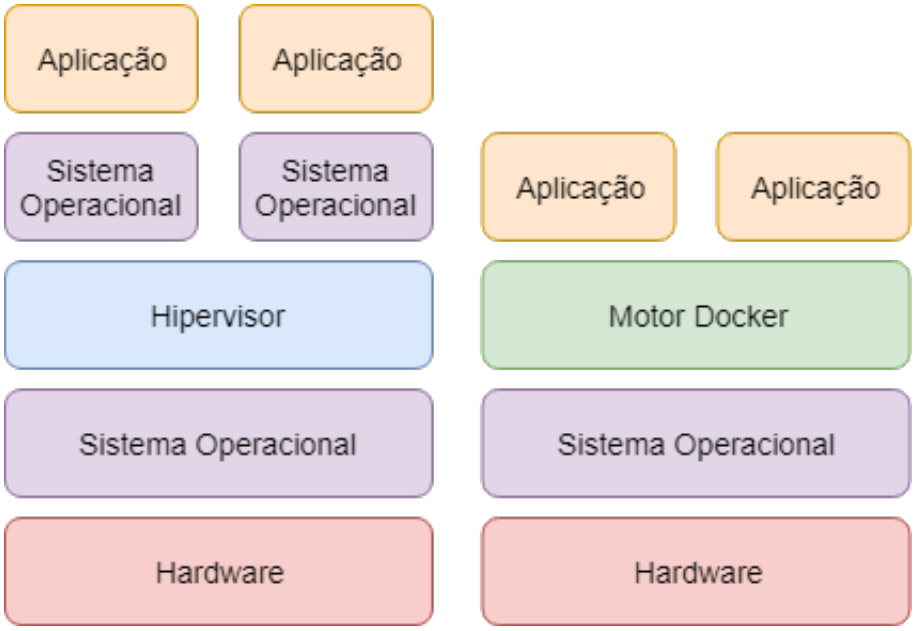

Figura 2. Arquitetura virtual vs. Arquitetura Docker

3.2. VirtualBox

O VirtualBox é, sem dúvidas, o *software* de virtualização mais conhecido. Por não depender de assinatura para uso, atrai os mais diversos públicos, desde entusiastas até empresas de pequeno e médio porte.

Foi desenvolvido, primeiramente, pela empresa alemã Innotek, sob uma licença para uso, ou seja, uma versão paga. Posteriormente, em meados de 2007, ocorreu o lançamento da versão OSE - Open Source Edition, sob licença GPL - *GNU General Publice License*. Em 2009 a Innotek foi comprada pela Sun Microsystems [Wire 2008] e, no ano seguinte, adquirida pela Oracle [Guardian 2009].

O VirtualBox utiliza a arquitetura virtual conforme mostrado na Figura 1. É extremamente intuitivo e fácil de usar, o que evidencia sua aceitação pelo público em geral.

3.3. VMware ESXi

Criado pela VMWare, o ESXi é um hipervisor corporativo utilizado em servidores de grande porte. Ao contrário do Docker, por exemplo, o ESXi não é um software aplicativo, ou seja, sua instalação ocorre diretamente nos componentes compuntacionais, excluindo a necessidade de um sistema operacional (Figura 3). Por ser integrado a nível de *hardware* problemas oriundos das máquinas virtuais tradicionais como demora no I/O, sobrecarga de disco e demais problemas mencionados anteriormente são mitigandos, tornando o ESXi uma alteranativa extremamente atraente para instituções que dependem de alta disponibilidade de serviços.

Ao ser instalado uma camada de virtualização é inserida entre o hardware e o sistema da VMware e, por meio de um serviço cliente, que pode ser instalado em qualquer computador pessoal, pode-se ter acesso a todas funcionalidades do ESXi, tal como a criação de máquinas virtualizadas para execução de diversos sistemas e aplicações.

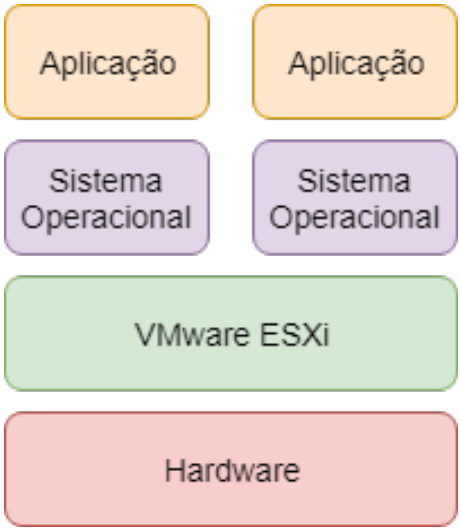

Figura 3. Arquitetura do VMware ESXi

3.4. Xen

Para entendermos melhor o que é o Xen, é necessário entender primeiramente o que é virtualização total e para-virtualização.

Diogo Menezes [Mattos 2008] apresenta uma definição sobre esses conceitos, uma virtualização total tem por objetivo fornecer ao sistema hospedado uma réplica do hardware sistema hospeiro. Como o sistema hospedado é executando sem modificações, este não tem conhecimento de que é um sistema virtualizado e, portanto, todas as ações executadas serão interpretadas primeiramente no sistema hospedado, depois no hipervisor e somente depois, no hardware. Deste modo, percebe-se que há uma sobreposição de execuções, o que ocasiona lentidão da máquina como pode ser visualizado na Figura 4.

Já em uma para-virtualização, o sistema que será instalado é modificado para compreender que é um sistema operacional para esse fim. Deste modo, o hipervisor é sempre chamado quando uma ação é feita nesse sistema, removendo uma troca de contexto e, portanto, ganhando performance.

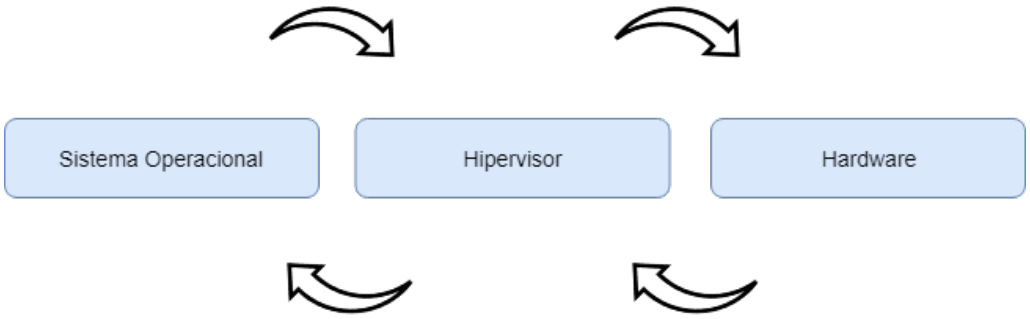

Figura 4. Fluxo de execução na virtualização total

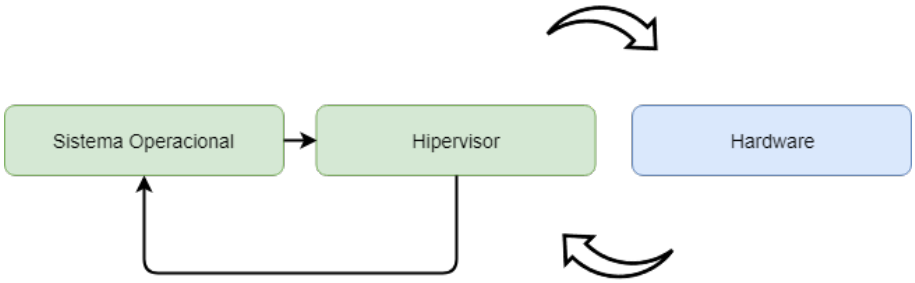

Figura 5. Fluxo de execução na para-virtualização

Desenvolvido originalmente na Universidade de Cambridge, na Inglaterra, o Xen é um dos mais populares para-virtualizadores do mercado. Foi adquirido pela Citrix System Inc. em 2007 [Inc. 2007].

4. Implementação

Desenvolvido pela AdaptiveScale, o LXDUI tem como objetivo principal disponibilizar uma interface amigável de gerencimaneto de um ambiente que utiliza LXC/LXD, visto que, nativamente, esses componentes não fornecem qualquer tipo de interface, a não ser, é claro, uma interface CLI - *Command Line Interface*.

Dentre as diversas funcionalidades disponíveis, podem-se citar as seguintes:

- Criação de usuários;
- *Download* de diversas imagens base, como Ubuntu, Debian e CentOS;
- Personalização de recursos quantidade de memória, armazenamento, processador etc;
- Criação de perfis de imagem.

4.1. Equipamento Utilizado

Para execução desse projeto foi utilizado uma máquina com processador Intel Core I5-7200 @ 2.50GHz * 4, 8 GB de memória, disco Samsung SSD 850 256GB, placa Gráfica Intel® HD Graphics 620 (Kaby Lake GT2) e sistema operacional Ubuntu 18.04.1 LTS.

4.2. Processo de Instalação

Para execução do projeto é necessário, primeiramente, instalar alguns componentes essenciais para o seu correto funcionamento.

Pyhon, versão 3: linguagem de programação de alto nível que abrange diversos paradgimas como, script, interpretado, orientado a objetos etc. Será utililzado como interpretador do projeto [Python].

Pip, versão 3: gerenciador de pacotes do Python. Mediante este é possível instalar diversos pacotes que podem ser utilizados durante o desenvolvimento.

Snap: gerenciador de pacotes de código aberto disponibilizador para diversas distribuições Linux, como o Ubuntu. Sua vantagem é o empacotamento de todas as suas dependências em um único *package*, simular ao .exe do Microsoft Windows [Snap].

Openssl: é um kit de ferramentas para protocolos TLS (Transport Layer Security) e SSL (Secure Sockets Layer) [Openssl].

Git: sistema de controle de versão amplamente utilizado, desde desenvolvedores independentes até grandes corporações [Git].

Para realizar a instalação dos requisitos necessários abra o interpretador de comandos Linux, como o *Bash* e execute os seguintes comandos:

```
$ sudo apt install git build-essential libssl-dev python3-venv
$ sudo apt install python3-dev zfsutils-linux bridge-utils
```

Uma vez que as bibliotecas necessárias estejam instaladas, é necessário instalar a versão estável no repositório Snap do Ubuntu:

```
$ sudo snap install lxd
```

Em seguida, é necessário baixar e extrair os arquivos do projeto e executar o comando para instalar a interface LXDUI modificada:

```
$ sudo python3 setup.py install
```

Isso encerra os passos de instalação da interface e das bibliotecas necessárias.

4.3. Desempenho

Afim de verificar a estabilidade e desempenho do LXDUI, verificou-se a quantidade de ambientes (máquinas) executadas virtualmente e qual o comportamento de memória e processador para manter tais ambientes ativos comparado com o VirtualBox.

Para a criação de 100 máquinas utilizando o sistema operacional Ubuntu 17.10 Minimal, o LXDUI gastou 4 minutos e 12 segundos. Esse conjunto de sistemas em execução consomem cerca de $0,7\%$ de memória, ou seja, um total de 57MB. Entretanto, cada nó criado consome em média $2,8$MB, totalizando 337MB todo o sistema. Também verificou-se o espaço em disco ocupado por cada container e cada um ocupa um total de 4KB, totalizando 400KB.

Como no VirtualBox a instalação é manual e as configurações são feitas em *background* conforme o usuário avança as etapas do processo, não é possível mensurar com exatidão o tempo total de instalação. Entretanto, apenas avançando essas etapas,

sem a preocupação com configurações personalizadas, como nome do usuário, nome da máquina etc, o processo levou um total de 15 minutos e 7 segundos para finalização. Já o espaço ocupado por cada máquina em disco é de $2,6$GB.

O processo foi executado três vezes visando refinar os resultados e estabelecer uma média de tempo e os seguintes foram obtidos:

Tabela 1. Tabela comparativa de testes de desempenho

	Teste 1	Teste 2	Teste 3	Média
LXDUI	00:04:12	00:05:32	00:04:58	00:04:54
VirtualBox	00:15:07	00:14:29	00:15:18	00:14:58

Após finalização dos testes, percebe-se que o LXDUI, mesmo que com 100 máquinas ativas, se mantém estável nos 337MB de consumo, com poucas oscilações no processamento. Já o VirtualBox ocasionou congelamento da máquina de testes após a décima máquina ficar ativa, sendo necessário reinicar a máquina de testes.

4.4. Criação de Usuário

Na criação de usuário, modificamos a funcionalidade para permitir que seja inserido um tipo de usuário, sendo possível assumir dois valores:

Tabela 2. Tabela comparativa de tipos de funções

Tipo	Função
Admin	Acesso a todas as funções do sistema
User	Acesso somente a criação de novos containers

```
root@marco-Inspiron-7560:/home/marco/lxdui-master# python3 run.py user add -u usuario1 -p usuario1
root@marco-Inspiron-7560:/home/marco/lxdui-master# python3 run.py user list
User Accounts:
1. admin
2. usuario1
```

Figura 6. Criação de usuário - Ates

```
root@marco-Inspiron-7560:/home/marco/tcc-lxdui# python3 run.py user add -u usuario1 -p usuario1 -t admin
root@marco-Inspiron-7560:/home/marco/tcc-lxdui# python3 run.py user list
User Accounts:
1. usuario1. admin
```

Figura 7. Criação de usuário - Depois

Caso o tipo do usuário seja omitido no momento da criação, o valor ”user”será considerado.

```
root@marco-Inspiron-7560:/home/marco/tcc-lxdui# python3 run.py user add -u usuario2 -p usuario2
root@marco-Inspiron-7560:/home/marco/tcc-lxdui# python3 run.py user list
User Accounts:
1. usuario1. admin
2. usuario2. user
```

Figura 8. Criação de usuário - Omissão do Tipo

4.5. Execução do Sistema

No desenvolvimento inicial, a inicilização do sistema sempre daria acesso a todas as funcionalides do sistema, dentre elas, por exemplo, a possibilidade do usuário X remover um container do usuário Y.

Pensando nessas possibilidades modificamos o comando de inicialização do para permitir um novo parâmetro, *Mode*. Esse parâmetro é responsável por permitir a visualização ou não a visualização de controles avançados, mitigando eventuais erros do usuário.

```
root@marco-Inspiron-7560:/home/marco/lxdui-master# python3 run.py start
LXDUI ver. 2.1.2 -- (c)AdaptiveScale, Inc.
http://www.adaptivescale.com
LXDUI started. Running on http://0.0.0.0:15151
PID=20924, Press CTRL+C to quit
```

Figura 9. Inicialização - CLI - Antes

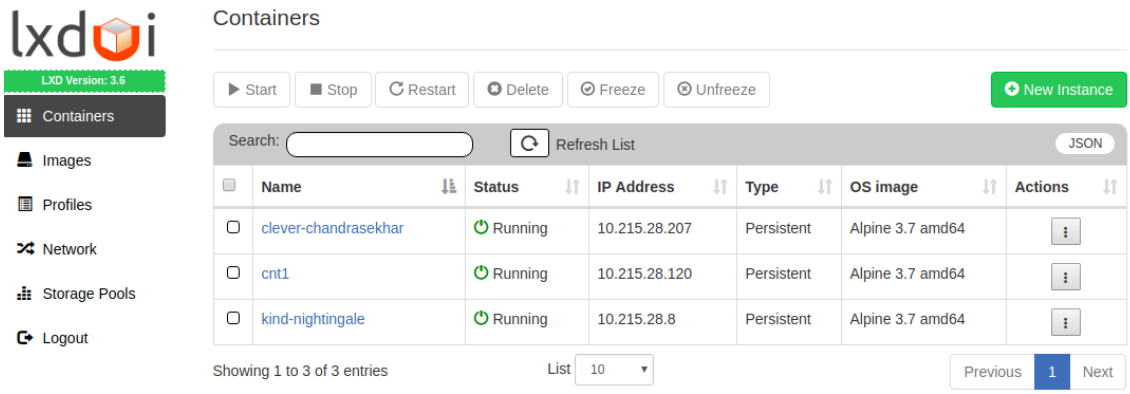

Figura 10. Inicialização - Web - Antes

```
root@marco-Inspiron-7560:/home/marco/tcc-lxdui# python3 run.py start -m admin
LXDUI ver. 2.1.2 -- (c)AdaptiveScale, Inc.
http://www.adaptivescale.com
LXDUI started. Running on http://0.0.0.0:15151
PID=21255, Press CTRL+C to quit
```

Figura 11. Inicialização - CLI - Depois

Assim como na criação de usuário, se o parâmetro que indica o modo de inicialização for omitido, o valor *user* será considerado e, portanto, removendo os controles avançados.

4.6. Configuração padrão

Ao inicializar o sistema em modo de usuário, não será possível alterar configurações do container, logo, na sua criação também não será. Deste modo essas configurações assumirão alguns valores padrão e só poderão ser modificados pelo administrador.

Para modificação dos valores é necessário alteração direta no código fonte da aplicação.

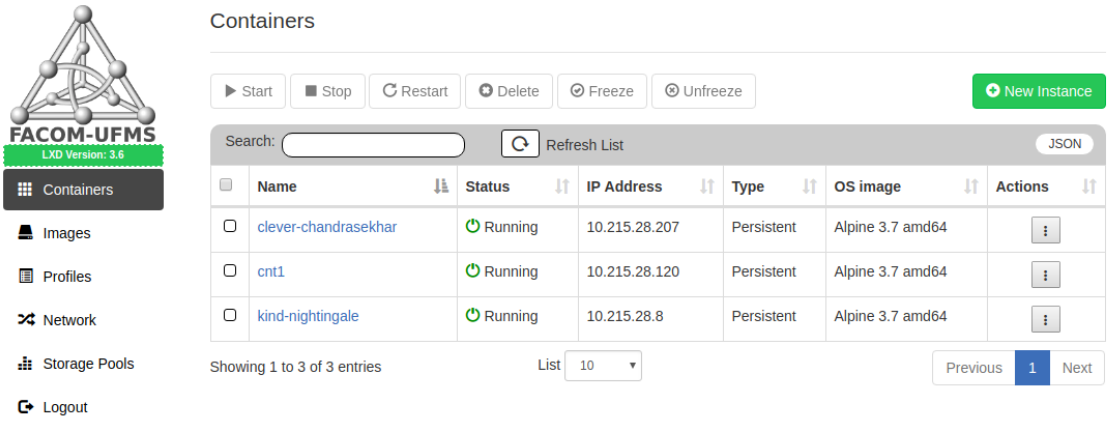

Figura 12. Inicialização - Web - Depois

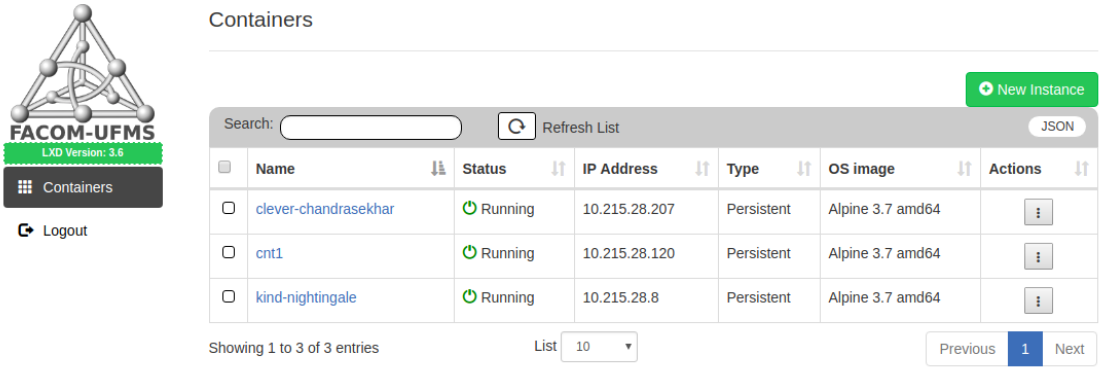

Figura 13. Inicialização - Web - Modo usuário

Tabela 3. Tabela de valores padrão para criação de container

Função	Valor
Persistent	True
Autostart	True
Memory	64
CPU Allocation	100
Hard Limit	True

5. Conclusão

Nota-se que o LXD é um sistema altamente escalável pois elimina a camada de virtualização e, por consequência, a troca de informações entre o sistema hospedeiro e o sistema hospedado é mais rápida. Entretanto sistemas auxiliares devem ser desenvolvidos, assim como o LXDUI, para que o gerenciamento dos recursos e informações dos containers e imagens seja facilitado.

Ainda que o projeto necessite de maiores ajustes quanto a segurança, isolamento total de usuários, esse se torna um projeto atraente para execução em ambientes de pesquisa e teste como, por exemplo, no acadêmico. Outro ponto que pode ser melhorado é a de autenticação de usuários. Atualmente é necessário que o administrador do LXDUI

cadastre os usuários um a um, entretanto é possível realizar a autenticação por meio de um servidor LDAP - *Lightweight Directory Access Protocol*, visto que, o framework web utilizado no projeto (Flask) possui bibliotecas para que tal funcionalidade seja implementada.

Por outro lado, alternativas como o Docker e o VirtualBox, ainda que execute um sistema totalmente virtualizado, podem ser utilizados com menos implementações pois já possuem tais camadas, porém, o custo extra de recurso no hospeiro deve ser levado em consideração.

5.1. Trabalhos futuros

A interface é funcional e permite que os usuários vejam apenas uma versão simplificada do LXDUI, diferente da parte administrativa que da acesso as configurações de disco, memória, processador, etc. Ainda é necessário implementar o controle de acesso por LDAP para que os usuários possam usar uma base centralizada de autenticação e também a limitação, via interface administrativa, de quantos containers cada usuário pode rodar no máximo, sem a possibilidade de visualizar os containers dos outros usuários.

Referências

AdaptiveScale (2018). A web UI for Linux containers based on LXD/LXC. `https://github.com/AdaptiveScale/lxdui`. [Online; accessed 20-July-2018].

Banerjee, T. (2014). Lxc vs lxd vs docker-making sense of the rapidly evolving container ecosystem.

Baukes, M. (2017). Docker vs LXC. `https://www.upguard.com/articles/docker-vs-lxc`. [Online; accessed 06-November-2018].

de Almeida, M. O. D. `https://git.facom.ufms.br/201519030525/tcc-lxdui`. [Online; accessed 18-November-2018].

Foundation, C. N. C. https://kubernetes.io/", year = 2018, note = "[online; accessed 20-july-2018]".

Git. `https://git-scm.com/about`. [Online; accessed 10-November-2018].

Guardian, T. (2009). Oracle's takeover of Sun Microsystems comes as surprise to software industry. `https://www.theguardian.com/business/2009/apr/20/sun-microsystems-oracle-takeover`. [Online; accessed 11-November-2018].

Inc., C. S. (2007). Citrix to buy virtualization company XenSource for $500 million. `https://www.cnet.com/news/citrix-to-buy-virtualization-company-xensource-for-500-million/`. [Online; accessed 29-Octomber-2018].

Mattos, D. M. F. (2008). Virtualização: VMWare e Xen.

Microsoft (2018). Hyper-V Virtualization. `https://docs.microsoft.com/pt-br/virtualization/hyper-v-on-windows/about/`. [Online; accessed 20-July-2018].

Oliveira, P. (2017). Kernel do Linux: O que é e para que serve? https://www.escolalinux.com.br/blog/kernel-do-linux-o-que-e-e-para-que-serve. [Online; accessed 06-November-2018].

Openssl. https://www.openssl.org. [Online; accessed 10-November-2018].

Pfaff, B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al. (2015). The Design and Implementation of Open vSwitch. In *NSDI*, volume 15, pages 117–130.

Python. https://www.python.org. [Online; accessed 10-November-2018].

Snap. https://en.wikipedia.org/wiki/Snappy_(package_manager). [Online; accessed 10-November-2018].

Vaughan-Nichols, S. J. (2014). Docker libcontainer unifies Linux container powers. https://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/. [Online; accessed 18-November-2018].

VMware (2018). VMware Enterprise. https://www.vmware.com. [Online; accessed 20-July-2018].

Wire, B. (2008). Sun Microsystems Announces Agreement to Acquire innotek, Expanding Sun xVM Reach to the Developer Desktop. https://www.businesswire.com/news/home/20080212005461/en/Sun-Microsystems-Announces-Agreement-Acquire-innotek-Expanding. [Online; accessed 11-November-2018].