Rufus: Ferramenta para o gerenciamento de infraestrutura para a execução de aplicações em containers

Jonatan Souza[1,2], **Allan Santos**[1,2], **Matheus Bandini**[1], **Henrique Klôh**[1], **Bruno Schulze**[1]

[1] Laboratório Nacional de Computação Científica
(LNCC) – 25.651-075, Petrópolis, RJ, Brazil

[2]Faculdade de Educação Tecnológica do Estado do Rio de Janeiro
(FAETERJ-Petrópolis) – Petrópolis, RJ, Brazil

{jgsouza,amsantos}@faeterj-petropolis.edu.br, {mbandini,henrique,schulze}@lncc.br

Abstract. *This article presents Rufus, a tool to enable users who have little or no knowledge on infrastructure management to be able to submit jobs to be performed in a computation environment suitable for their application. This is achieved through the application submission and management in a cloud computing based on the container technology. With Rufus, one can create dedicated environments with specific configurations to meet the requirements of each application being executed. Authentication and management of users and files are also supported by Rufus.*

Resumo. *Este artigo apresenta a ferramenta Rufus, cujo objetivo é permitir que usuários que detenham pouco ou nenhum conhecimento sobre gestão de infraestrutura, sejam capazes de submeter trabalhos para serem executados em um ambiente computacional adequado para a sua aplicação. Isso se dá através da submissão e da gerência das aplicações em uma Nuvem Computacional baseada na tecnologia de Containers. Através da ferramenta, é possível criar um ambiente dedicado, com configurações personalizadas de acordo com as necessidades de cada aplicação a ser executada. Além disso, o Rufus oferece recursos como autenticação, gerência de usuários e gerência de arquivos.*

1. Introdução

O uso da computação científica permitiu o avanço dos demais campos da ciência com o desenvolvimento de pesquisas baseadas em simulações. Contudo, muitos centros de pesquisas ainda dependem da habilidade individual dos cientistas para o uso eficiente dos recursos computacionais necessários para realizar experimentos. Sem o auxílio de uma ferramenta que facilite a construção de *appliances* de aplicações e apoie a execução de *workflows*, o experimento pode ficar sujeito a falhas.

Um *workflow* científico pode ser definido como a descrição formal de um processo científico, dividido em passos, a serem executados em um determinado experimento [Deelman et al. 2009]. As aplicações científicas, que realizam cada um desses passos, necessitam, em geral, de ambientes configurados com dependências e ajustes específicos. As configurações necessárias para cada uma dessas aplicações podem ser completamente diferentes. Portanto, o compartilhamento de um mesmo ambiente por diversas aplicações pode gerar resultados indesejáveis para os gestores do ambiente.

A evolução da tecnologia de virtualização de recursos computacionais tornou-se extremamente útil, no tocante às propriedades dos *workflows* científicos, tendo em vista a complexidade de configuração das aplicações [Deelman et al. 2009] e dos recursos nos quais elas são executadas. O uso da virtualização permitiu o encapsulamento das aplicações em ambientes com recursos exclusivos, de forma a simplificar sua utilização voltada para sistemas distribuídos. Além disso, os ambientes virtualizados podem ser replicados na infraestrutura com o objetivo de aplicar o conceito DRY (*Don't Repeat Yourself*) e de facilitar a manutenção.

Recentemente, a virtualização em nível de sistema operacional é uma das formas de virtualização que mais cresce em popularidade. Isso, porque oferece um desempenho próximo do que é oferecido pelo *baremetal*. Como exemplo, pode-se citar o uso de *containers*, que é um método de virtualização que permite múltiplas instâncias de sistemas GNU/Linux executadas em um único *host*, enquanto dividem o mesmo *kernel* [Lamps et al. 2014]. Desta forma, a utilização de *containers* para a criação de *clusters* virtuais apresenta-se como uma alternativa efetiva e de baixo custo.

Este trabalho apresenta uma ferramenta, chamada Rufus, que foi desenvolvida visando aplicar o conceito de virtualização em nível de sistema operacional provendo *appliances* de aplicações através do LXC [1] (*LinuX Containers*) como infraestrutura para a execução de *workflows* científicos. Também é disponibilizada uma interface gráfica para a execução desses *workflows* em uma linguagem de alto nível com cliques e arrastes de *mouse*. Dessa forma, espera-se facilitar, para pesquisadores, as tarefas relacionadas ao gerenciamento e à replicação dos ambientes para a execução das aplicações. A ferramenta Rufus está dividida em dois níveis: Rufus-core e Rufus-web. O primeiro é responsável por realizar o gerenciamento dos recursos computacionais e o segundo é responsável pela interface gráfica e pela interação com o usuário.

A demonstração da ferramenta será realizada em duas etapas: a primeira envolve a criação de um *appliance* com as configurações necessárias para a execução de uma aplicação, enquanto a segunda consiste em submeter uma tarefa usando o *appliance* configurado através do módulo *Composer* do Rufus-web. A descrição completa dos procedimentos realizados na demonstração foram omitidas deste documento por limitação de espaço. No endereço `http://rufus.lncc.br/manual` é possível encontrar um tutorial passo-a-passo que guia o usuário através da instalação, configuração e uso da ferramenta. Na Seção 2 são apresentadas algumas ferramentas que se relacionam com o desenvolvimento ou o objetivo deste trabalho. A Seção 3 apresenta a ferramenta proposta. A Seção 4 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

Ambientes utilizados para a execução de aplicações científicas são, em geral, compostos de máquinas com alta capacidade de processamento e de armazenamento, podendo haver o compartilhamento de tais recursos por diversas aplicações. Para evitar conflitos entre as diferentes aplicações e suas diferentes versões e permitir a utilização do ambiente de uma forma mais granular, é comum, segundo [Vogels 2008], isolar cada aplicação em um ambiente virtualizado. Esta seção descreve alguns aspectos de ferramentas que foram desenvolvidas com o objetivo de utilizar a virtualização para este fim.

[1] https://linuxcontainers.org/

O Juju [2] é um sistema orquestrador de serviços em nuvem desenvolvido pela Canonical Ltd. que também é a empresa que desenvolve o Ubuntu GNU/Linux. No contexto deste trabalho, as principais características do Juju são:

- O isolamento das aplicações pode ser feito através de máquinas virtuais ou *containers*;
- Um ambiente gráfico para a modelagem do ambiente;
- Ambiente multiusuário.

O Juju realiza o *deploy* de serviços em um ambiente de nuvem, além de permitir que as aplicações sejam isoladas em diferentes ambientes. Com o Juju é possível a utilização do OpenStack, do MaaS [3] (*Metal as a Service*) ou do LXC para o gerenciamento dos ambientes. A interface gráfica do sistema representa as aplicações com ícones em um *canvas* e as relações entre essas aplicações através de conexões que o usuário pode desenhar entre os ícones.

No Rufus, as conexões entre as aplicações representam a direção em que os dados gerados como resultado de uma podem ser passados como entrada para a próxima. As dependências entre as aplicações definem a ordem de execução, formando assim um *workflow*. O que se torna uma vantagem em relação ao Juju, pois este não oferece tal funcionalidade.

Docker [Boettiger 2015] e LXD [4] são duas ferramentas utilizadas para o gerenciamento de instâncias virtuais na forma de *containers* em um nível mais elevado de abstração do que o LXC. Ambas as ferramentas podem ser utilizadas através de API REST. Uma característica interessante do Docker é que ele é capaz de utilizar *containers* em sistemas operacionais diferentes de GNU/Linux. Além disso, no OpenStack também é possível utilizar o LXD [Short 2015] e o Docker [Fifield et al. 2014].

A solução utilizada neste trabalho para o gerenciamento de recursos procura ser mais simples do que as ferramentas citadas, no tocante a integração com os sistemas que possam se beneficiar do isolamento oferecido pela tecnologia de *containers*, como o portal de submissão e gerenciamento de aplicações descrito neste trabalho. Além de oferecer *containers* que, ao mesmo tempo, possam ser utilizados como uma instância da aplicação científica instalada, e permaneçam tão flexíveis quanto os criados diretamente pelo LXC.

Embora o uso de *containers* esteja amplamente difundido atualmente, muitas pesquisas científicas deixam de se beneficiar desta tecnologia, pois os seus experimentos são executados por alguma ferramenta de *workflow* científico sem algum suporte que automatize o gerenciamento de *containers*. O trabalho apresentado neste artigo propõe uma nova ferramenta de *workflow* científico que considera o uso de *containers* desde o início do seu desenvolvimento com o objetivo de obter a melhor integração possível entre as tecnologias.

Entretanto, existem também pesquisas sobre a adaptação de ferramentas de *workflow* científico existentes como [Zheng and Thain 2015] e [Gerlach et al. 2014]. O primeiro adapta as ferramentas Makeflow e WorkQueue para utilizar o Docker com o ob-

[2] http://www.ubuntu.com/cloud/juju
[3] http://www.ubuntu.com/cloud/maas
[4] https://linuxcontainers.org/lxd/

jetivo de encapsular as aplicações de *workflow*. Esta adaptação garante a reprodutibilidade dos experimentos eliminando as diferenças entre ambientes distintos que poderiam causar erros na execução de um *workflow*. O segundo apresenta o projeto Skyport, que também utiliza o Docker para adaptar o AWE/Shock com o objetivo de solucionar problemas de *deployment* e ampliar a eficiência da utilização de recursos.

3. Descrição da Ferramenta

A arquitetura da ferramenta descrita por este trabalho está divida em dois níveis: Rufus-web e Rufus-core, que podem ser observadas na Figura 1. Juntos, eles têm por finalidade oferecer as seguintes funcionalidades:

- Prover um controle de usuário divido por hierarquia;
- Proporcionar a gerência dos recursos computacionais através de uma interface gráfica;
- Permitir que o usuário crie e configure *appliances*. O *appliance* será considerado neste trabalho como uma instância que atenda as necessidades da aplicação;
- Prover um ambiente onde usuários possam submeter aplicações, utilizando *appliances* pré-configurados, através do próprio navegador.

Para oferecer essas funcionalidades, foi desenvolvido um portal *web* que somado a um motor gerador de *containers*, disponibiliza um ambiente completo ao pesquisador, com os objetivos de automatizar tarefas e facilitar a configuração da infraestrutura. O portal *web* provê uma interface para a gerência de recursos computacionais, autenticação de usuários, armazenamento de dados e a preparação e submissão de aplicações.

A gerência de infraestrutura se torna complexa a medida que a demanda por recursos computacionais aumenta. Além disso, a forma como esses recursos são oferecidos pode variar. Considerando essas características, as tarefas de gerenciamento da infraestrutura foram designadas a um componente independente, denominado Rufus-core. O Rufus-core é um motor gerador de *containers* e é responsável pela infraestrutura e por realizar as tarefas de gerência de recursos descritas no portal. Através dele, as aplicações são executadas.

3.1. Rufus-Core

O Rufus-core [Santos et al. 2015] é um motor gerador de ambientes virtualizados responsável pela criação de *containers* e por disponibilizá-los na forma de *appliances* pré-configurados dedicados às aplicações científicas. O objetivo é prover uma camada, acessível por *API REST*, a fim de atender aos requisitos das aplicações dos pesquisadores, criando dinamicamente ambientes para executar aplicações.

Os *containers* são criados através do LXC e as aplicações devem ser instaladas manualmente pelo administrador durante o processo de criação do *appliance*. Se o *container* oferecer o serviço *shell-in-a-box* [5], é possível requisitar uma URL ao Rufus-core para ter acesso a um terminal via navegador. Através deste, o administrador pode realizar remotamente qualquer instalação e configuração desejada no *container*.

O Rufus-core está dividido em dois módulos: i) `webservice` que implementa a *API REST* e realiza a comunicação com o portal Rufus-web; ii) `libpylxc` que realiza as

[5]https://code.google.com/p/shellinabox/

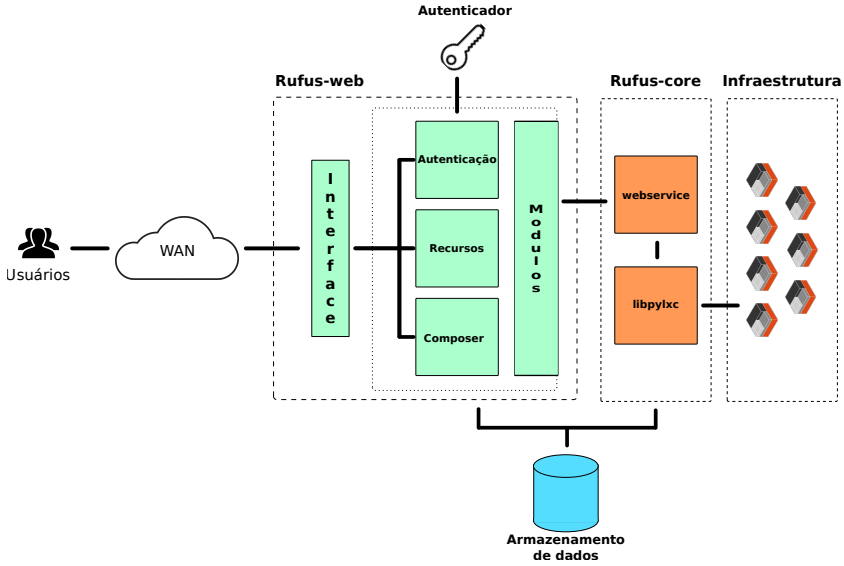

Figure 1. Visão geral da arquitetura da ferramenta Rufus.

operações necessárias para possibilitar as funcionalidades que o Rufus-core possui, como ilustrado na figura 1. O desenvolvimento desses módulos é feito em linguagem Python3, foi utilizado o *framework* Flask para o desenvolvimento do *webservice* e o formato JSON para a serialização das requisições feitas para a *API*. As funcionalidades disponíveis na API estão listadas a seguir:

- Listar, filtrar ou exibir em detalhes os *containers* existentes e também listar os *templates* e imagens do LXC.
- Mostrar recursos disponíveis na máquina real;
- Criar, clonar, alterar o estado e destruir *containers*;
- Disponibilizar *appliances* de aplicações científicas criados através de *containers* pré-configurados; e
- Executar aplicações em paralelo.

Para executar uma aplicação, é feito um clone do *container* no qual a aplicação foi instalada utilizando CoW (*Copy on Write*). Dessa forma, o desempenho do processo de clonagem não é prejudicado caso a aplicação seja muito grande, pois apenas referências aos arquivos originais são criadas. Os arquivos do *container* só são realmente copiados a medida que são modificados. Isso torna mais eficiente a criação de uma nova instância de uma aplicação. Para permitir o uso deste tipo de clonagem, foi utilizado o sistema de arquivos em camadas, chamado *OverlayFS* [6] que faz parte do *kernel Linux* nas versões recentes. Há também um módulo `client` que utiliza a biblioteca `requests` que foi desenvolvido com o objetivo de facilitar o uso da Rufus-core por *scripts* feitos em Python3, automatizando a criação das requisições HTTP.

3.2. Portal Rufus-web

O portal *web* Rufus tem o objetivo de prover ao usuário a interação com o sistema através de uma interface amigável. O portal está dividido em quatro módulos: **i)** Autenticação, **ii)** Gerência de Recursos, **iii)** Armazenamento de Dados e **iv)** Composer.

[6]https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

Tendo em vista a segurança dos dados e também da rede na qual os recursos computacionais estão hospedados, foi implementado um módulo de autenticação baseado no sistema *OAuth2* [Hammer-Lahav and Hardt 2011]. Isso garante que apenas usuários autenticados no sistema poderão acessar as páginas do portal.

O Portal *web* divide os usuários em dois níveis:

- Administrador: usuários com esse privilégio têm acesso a todas as funcionalidades no que tange à gerência de recursos computacionais, sendo possível criar *appliances* e acessá-los via navegador através de um terminal *shell* e configurar as aplicações. Este tipo de usuário também conta com uma área na qual é possível tornar usuários comuns em administradores através do cadastro do e-mail;
- Pesquisador: este tipo de usuário é capaz apenas de elaborar e executar aplicações utilizando os *appliances* predefinidos pelos administradores.

O objetivo de apenas administradores terem acesso direto aos recursos computacionais é manter um número restrito de usuários com acesso direto aos *appliances* criados, aumentando assim a segurança do portal.

Com o objetivo de ampliar o controle sobre a infra-estrutura foi desenvolvido um módulo para a gerência de recursos. Neste módulo, através de uma interface gráfica intuitiva, administradores podem criar um *appliance* com o nome desejado, com um sistema *GNU/Linux* com diversas distribuições providas pelo Rufus-core. O módulo ainda provê um acesso remoto através de um terminal *shell* emulado no próprio navegador, através do qual é possível configurá-lo com as diretivas desejadas para a aplicação. Todos os *appliances* configurados pelo administrador estarão disponíveis para uso na área de submissão de *workflows*.

Para exportar os dados armazenados foi utilizado o NFS (*Network File System*) que é um sistema de arquivos distribuídos, a fim de compartilhar arquivos e diretórios entre computadores conectados em rede, formando um diretório virtual. Sendo assim, fica disponível ao Rufus-core as informações necessárias para a execução das tarefas.

No primeiro acesso ao portal Rufus, é criado um diretório com um identificador exclusivo para cada usuário. O e-mail obtido pelo autenticador é usado para nomear o diretório, tornado-o único e rastreável. Essa estrutura é importante, pois garante que cada usuário terá acesso apenas ao seu diretório. Ao fazer *upload* de um arquivo no portal, estes são salvos no subdiretório `files`. Após submetido um *workflow*, é criado um subdiretório com o nome do *workflow*. O Rufus-core, após a execução de cada *workflow* submetido, armazena os resultados. Esses arquivos podem ser obtidos através de *download*.

O módulo Composer permite ao pesquisador elaborar um *workflow* em um quadro de desenhos utilizando uma linguagem de alto nível com cliques e arrastes de *mouse*. Todas as instâncias configuradas pelos administradores ficam disponíveis na forma de *appliances* para o pesquisador no menu lateral. Cada *workflow* ao ser submetido deve ter no mínimo um arquivo e um *appliance*.

Quando um *workflow* é submetido, a requisição vai para o controlador de *workflows* do portal, que é responsável pela preparação para a submissão. O primeiro passo é organizar as informações dividindo em níveis (Figura 2), considerando suas dependências.

Cada nível então terá uma quantidade específica de aplicações a serem processadas, formando assim conjuntos de tarefas. Uma vez organizado, o controlador envia esse conjunto de tarefas para o Rufus-core de forma paralela. O controlador, então, aguarda o fim da execução deste conjunto de tarefas. Uma vez executado, o próximo conjunto é enviado para o gerente de recursos. Este ciclo se repete até alcançar o ultimo nível. Caso ocorra algum erro durante este processo o *workflow* é interrompido.

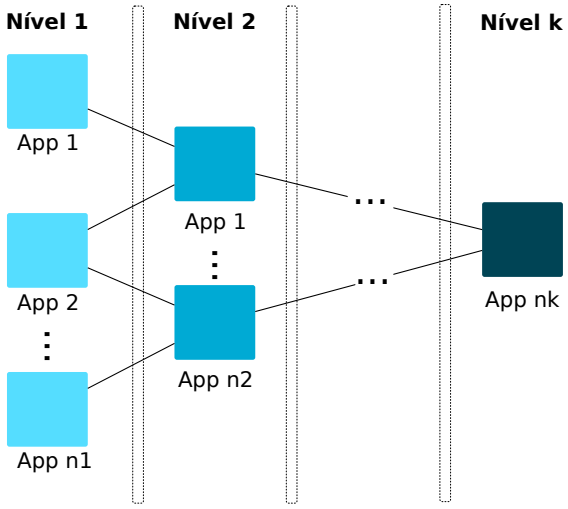

Figure 2. Exemplo de um *workflow* com K níveis

4. Conclusão e Trabalhos Futuros

O Portal *Rufus-web* permite a configuração de *appliances* e gerência de recursos computacionais, facilitando a submissão de *workflows* através de uma interface gráfica, considerando que as mesmas seriam tarefas onerosas quando realizadas sem o auxílio de uma ferramenta. A área de submissão de aplicações, (*Composer*), através do qual o pesquisador pode elaborar seu *workflow* ou aplicação e submetê-lo, é o grande diferencial e provê significativa contribuição à comunidade científica, tendo em vista a simplificação para a submissão dessas tarefas utilizando os *appliances* configurados com as aplicações.

Atualmente, o portal conta apenas com um gerente de recursos, denominado *Rufus-core*. Com a diversidade de infraestruturas de nuvens disponíveis, como por exemplo o *OpenStack* [Sefraoui et al. 2012] e o *OpenNebula* [Milojičić et al. 2011], a ideia é que o portal *Rufus-Web* proporcione uma gerência de *templates* capaz de integrá-las. Isso será possível com a implementação de interfaces que atendam às especificações de cada uma delas. Uma vez integradas, também será necessário a criação de uma inteligência capaz de escalonar a tarefa submetida, com o propósito de executá-la na infraestrutura que melhor se encaixe no perfil da aplicação, considerando que o portal comportará mais de um gerente de recursos.

Para que seja possível essa modularização, o Portal *Rufus-web* será divido em duas partes, sendo que a primeira possui o objetivo de permitir a interação e a autenticação de usuários, e uma segunda denominada *Rufus-AOD(AppFlow Organizer and Dispatcher)*. Esta última é responsável pela implementação das particularidades de cada gerente de recursos, além de realizar as requisições para execução das aplicações.

References

Boettiger, C. (2015). An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79.

Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540.

Fifield, T., Fleming, D., Gentle, A., Hochstein, L., Proulx, J., Toews, E., and Topjian, J. (2014). *OpenStack Operations Guide*, chapter 4. "O'Reilly Media, Inc.".

Gerlach, W., Tang, W., Keegan, K., Harrison, T., Wilke, A., Bischof, J., D'Souza, M., Devoid, S., Murphy-Olson, D., Desai, N., et al. (2014). Skyport: container-based execution environment management for multi-cloud scientific workflows. In *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*, pages 25–32. IEEE Press.

Hammer-Lahav, D. and Hardt, D. (2011). The oauth2. 0 authorization protocol. Technical report, IETF Internet Draft.

Lamps, J., Nicol, D. M., and Caesar, M. (2014). Timekeeper: a lightweight virtual time system for linux. In *Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of advanced discrete simulation*, pages 179–186. ACM.

Milojičić, D., Llorente, I. M., and Montero, R. S. (2011). Opennebula: A cloud management tool. *IEEE Internet Computing*, (2):11–14.

Santos, A. M. M., Kloh, H. M., Barbosa, J. P., and Schulze, B. (2015). Gerenciamento do motor gerador de containers para nuvens computacionais. In *ANAIS do XVI Workshop de Iniciação Científica em Arquitetura de Computadores e Computação de Alto Desempenho (WSCAD-WIC 2015)*. WSCAD-WIC.

Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42.

Short, C. (2015). Introduction to nova-compute-lxd. `https://insights.ubuntu.com/2015/05/06/introduction-to-nova-compute-lxd/`. [Online; accessed 01-February-2016].

Vogels, W. (2008). Beyond server consolidation. *Queue Virtualization Magazine*, 6(1):20–26.

Zheng, C. and Thain, D. (2015). Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '15, pages 31–38, New York, NY, USA. ACM.