UTILIZAÇÃO DE CONTAINERS: UM CONTEXTO HISTÓRICO

GISELE MARIA FREIRE COSTA[1]
CLAYTON EDUARDO DOS SANTOS[2]

RESUMO

A implantação de *datacenters* em infraestrutura própria, também denominada *on-premises*, requer investimentos de grande vulto — classificados como *CapEx* ou *Capital Expenditure* já no início das operações. Alternativas ao modelo de infraestrutura *on-premises* têm sido a cada dia mais utilizadas no cotidiano das empresas, em especial a computação em nuvem. Independentemente da tecnologia utilizada na implantação da infraestrutura — *on-premises*, em nuvem ou híbrida, as fragilidades permanecem semelhantes, tendo em vista que os serviços básicos utilizados são os mesmos: computação, armazenamento e transferência de dados. A tecnologia de virtualização de servidores está enraizada em grande parte das empresas com alguma relação ao segmento de tecnologia da informação. Apesar de relevante e plenamente funcional, sua adoção em ambientes de produção onde a alta disponibilidade e a elasticidade eventualmente venham a ser as regras de negócio mais importantes, é questionável no que se refere a performance, manutenção e padronização de ambientes. Nesse sentido, a utilização de *containers* apresenta vantagens relevantes em todos os aspectos supracitados. A proposta do presente trabalho é apresentar um contexto histórico de modo a demonstrar a utilização de *containers* como uma alternativa viável para substituir, de modo significativo, soluções tradicionalmente baseadas em virtualização.

Palavras-chave: alta disponibilidade; computação em nuvem; container; elasticidade; virtualização.

[1] Pós-graduanda em Gestão Estratégica de Tecnologia da Informação, Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - campus Bragança Paulista, e-mail: gifreirecosta@gmail.com
[2] Professor Doutor, Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – campus Bragança Paulista, e-mail: claytones@ifsp.edu.br

USE OF CONTAINERS: A HISTORICAL CONTEXT.

ABSTRACT

The implementation of datacenters in its own infrastructure, also known as local, requires large investments — classified as CapEx or Capital Expenditure at the beginning of operations. Alternatives to the on-premises infrastructure model have been increasingly used in the daily lives of companies, especially the cloud computing. Regardless of the technology used in the implementation of the infrastructure — local, cloud or hybrid, with similar weaknesses, considering that the basic services used are the same: processing, storage and data transfer. Server virtualization technology is ingrained in most companies with some relation to the information technology segment. Despite being relevant and functionally functional, its adoption in production environments where high availability and elasticity eventually become the most important business rules, is questionable when it comes to performance, maintenance and standardization of environments. In this sense, the use of containers presents relevant advantages in all the aforementioned aspects. The purpose of this work is to present a historical context in order to demonstrate the use of containers as a viable alternative to significantly replace traditional solutions based on virtualization.

Keywords: *high availability; cloud computing; container; elasticity; virtualization.*

1. INTRODUÇÃO

A implantação de *datacenters* em infraestrutura própria, também denominada *on-premises*, requer investimentos de grande vulto - classificados como *CapEx* ou *Capital Expenditure* já no início das operações, bem como investimentos parciais para atualização do parque computacional, em geral a cada três ou quatro anos, a depender das regras de negócio envolvidas em cada caso. Além das despesas relacionadas a aquisição de equipamentos voltados aos serviços de computação, armazenamento e interconectividade, vários outros custos diretos e indiretos devem ser levados em consideração, como equipe técnica especializada, *links* de *Internet*, locação imobiliária, energia elétrica, refrigeração, *racks*, cabeamento, bem como redundância de links, serviços de missão crítica e demais insumos necessários para os equipamentos de interconectividade.

Nesse sentido, a adoção de tecnologias que possibilitem maior aproveitamento dos serviços tidos como básicos, à saber: processamento, armazenamento e tráfego de dados, são de fundamental importância para que investimentos realizados previamente sejam preservados com vistas à plena utilização dos recursos provisionados, desde que dentro de uma margem de segurança pré-estabelecida e considerando possibilidades de expansão futura dentro do mesmo modelo ou considerando soluções híbridas.

Alternativas ao modelo de infraestrutura *on-premises* têm sido a cada dia mais utilizadas no cotidiano das empresas, dentre as opções disponíveis, destaca-se em especial o paradigma de computação em nuvem. A migração para o modelo de nuvem possui diferentes motivações, mas a utilização do modelo de investimento baseado em *OpEx* - ou *Operational Expenditure,* em substituição *ao CapEx* é um dos mais recorrentes e traz como efeitos colaterais positivos, velocidade no processo de implantação inicial de infraestrutura, sem a necessidade de investimento de capital antecipado, bem como agilidade em processos de tomada de decisão em momentos estratégicos.

Dentro dessa perspectiva, recursos de tempo e dinheiro outrora investidos em infraestrutura própria, podem ser revertidos em melhoria das condições de trabalho dos colaboradores, bem como dos produtos e serviços oferecidos aos clientes.

1.1. Motivação

Independentemente da tecnologia utilizada na implantação da infraestrutura — *on-premises*, em nuvem ou híbrida, em geral as fragilidades relacionadas à eventual ociosidade dos equipamentos permanecem semelhantes, tendo em vista que os serviços básicos utilizados são os mesmos: computação, armazenamento e transferência de dados.

Para embasar tal afirmação, tomemos como exemplo algumas das tecnologias utilizadas nos serviços de computação. Em qualquer um dos cenários supracitados, não são raras arquiteturas que utilizam soluções baseadas em virtualização, *containers* ou mais recentemente em computação *serverless* – modalidade também denominada como *Function as a Service* ou *FaaS*, bem como a eventual combinação destas.

Nesse contexto, as duas tecnologias mais empregadas na especificação de arquitetura em aplicações legadas ou em processo de migração para a nuvem são respectivamente, a de virtualização - em geral aplicada na modalidade "*lift and shift*" e a de *containers,* normalmente aplicada em cenários onde as culturas *agile*, *devops* e de micro serviços já tenham sido implementadas. Aplicações que fazem uso da modalidade *Function as a Service* em geral são *cloud native* que, apesar de tendência, não refletem a totalidade de projetos existentes.

1.2. Justificativa

A tecnologia de virtualização de servidores está enraizada em grande parte das empresas que, em maior ou menor escala, tem alguma relação com o segmento de tecnologia da informação. Apesar de relevante e plenamente funcional, sua adoção em ambientes de produção onde a alta disponibilidade e a elasticidade eventualmente

venham a ser as regras de negócio mais importantes, é questionável no que se refere ao ciclo de vida, performance, manutenção e padronização de ambientes. Nesse sentido, a utilização de *containers* apresenta vantagens relevantes em todos os aspectos supracitados.

1.3. Objetivo

A proposta do presente trabalho é apresentar um referencial teórico baseado em contextos históricos que, direta ou indiretamente, motivaram o surgimento de tecnologias de computação modernas, como o desenvolvimento ágil, a cultura *DevOps*, a alta disponibilidade, a elasticidade e a arquitetura de micro serviços. Nesse contexto, existe todo um enredo construído de modo a demonstrar que a utilização de *containers* é uma alternativa viável para substituir, de modo significativo, soluções tradicionalmente baseadas em virtualização, incluindo paradigmas mais recentes, como o de computação em nuvem.

2. CONTEXTUALIZAÇÃO TEÓRICA

2.1. Equipamentos de grande porte - Mainframes

Desde os primórdios da informática, tecnologias voltadas à otimização e melhor aproveitamento dos recursos computacionais de equipamentos de grande e pequeno porte, tem sido objeto de estudo de grandes empresas e pesquisadores da área.

Na era dos *mainframes*, a aquisição dos equipamentos de grande porte estava ao alcance de poucos – em geral empresas líderes do segmento e universidades, sobretudo em função do alto custo envolvido. Além disso, o interesse pela tecnologia também esbarrava nas limitações de uso referentes ao efetivo acesso aos computadores, tendo em vista que os sistemas operacionais utilizados nos equipamentos da época permitiam a execução de somente uma tarefa - realizada por um único usuário, a cada vez. Em face do exposto, a existência de grandes filas de espera para agendamento de utilização dos equipamentos era algo tido como normal

e esperado. Tal restrição era tão efetiva que, um simples erro na compilação de um código fonte mal escrito, implicava em um novo agendamento, que poderia levar dias ou até mesmo semanas para acontecer.

Motivados por esse tipo de demanda, um consórcio de instituições formado pelo *MIT - Massachusetts Institute of Technology*, a divisão de produtos para grandes computadores da *GE - General Electric* e a companhia de telefonia *AT&T Bell Laboratories* deram início ao projeto *MULTICS - Multiplexed Information and Computing Service*, um sistema operacional inicialmente conceitual que tinha a ambição de implementar na época tecnologias modernas que permanecem em uso até os dias atuais, em especial: multiprocessamento simétrico, definição de um sistema de arquivos hierárquico, disponibilidade de uma *userland* repleta de bibliotecas, compiladores e utilitários diversos, bem como um sistema de endereçamento de memória inovador. No entanto, os recursos computacionais disponíveis e atrelados ao projeto em um primeiro momento revelaram-se insuficientes para as pretensões almejadas. No período entre 1968 e 1969 a *Bell Laboratories* retirou-se lentamente do projeto por não acreditar que três instituições com objetivos de negócio completamente diferentes pudessem alcançar um resultado comum satisfatório a todas elas.

No mesmo ano, *Ken Thompson* e *Dennis Ritchie* começaram a trabalhar por conta própria em um sistema que denominaram *UNICS* — fazendo uma clara menção ao sistema conceito em que foi baseado, e que mais tarde seria chamado de *UNIX*, como ficou mundialmente conhecido. O sistema, originalmente escrito em *assembly* para a arquitetura *DEC PDP-7* da *Digital Equipment Corporation*, implementava de fato os principais recursos previstos no projeto inicial e tornou-se um grande sucesso rapidamente, em especial, por possibilitar o acesso multiusuário a um sistema operacional legitimamente multitarefa, possibilitando dessa forma um aproveitamento muito mais efetivo dos equipamentos. Tais recursos passaram a interessar a outros fabricantes de *mainframes* e à própria *DEC* que pensava em utilizar o sistema em seus novos modelos mais modernos e performáticos. No entanto, por ser originalmente escrito em linguagem de máquina, seriam necessários *ports* específicos do *UNIX* para cada uma das eventuais arquiteturas suportadas, algo absolutamente

inviável do ponto de vista de desenvolvimento e manutenção. Nesse sentido, após muito pensar, *Dennis Ritchie* resolveu escrever o código fonte do *UNIX* em uma linguagem de programação de alto nível, criada por ele especificamente para esse propósito, bastando para tanto a implementação de um compilador destinado a cada uma das arquiteturas eventualmente suportadas, de modo a viabilizar os *ports* a partir de um código fonte comum — estratégia muito mais inteligente e factível. Nascia assim a Linguagem C.

Posteriormente, o *UNIX* original da *AT&T* inspirou o surgimento de novas variantes, também voltadas a equipamentos de grande porte. Dentre os sistemas em questão, destacava-se em especial o *BSD – Berkeley Software Distribution*, variante originada na comunidade acadêmica da Universidade de *Berkeley* — Califórnia, que possibilitou o surgimento de novas contribuições significativas que permanecem disponíveis até os dias atuais nos chamados “sistemas operacionais modernos”.

2.2. A era da computação pessoal

Com o advento da computação pessoal ocorrido no final da década de 70, “cidadãos comuns” passaram a ter oportunidade e condições de adquirir um microcomputador pessoal – ou simplesmente PC, no conforto de suas residências, ou ainda, em pequenos e médios negócios. O tamanho e o preço dos equipamentos diminuíram drasticamente, no entanto, assim como ocorrido no chamado “grande porte”, passaram-se anos até que as soluções de sistemas operacionais disponíveis para estes dispositivos passassem a contemplar recursos e funcionalidades mais robustas e tidas como essenciais pelos profissionais da área, dentre as quais destacam-se: suporte multiusuário e processamento multitarefa.

Durante vários anos, o sistema operacional mais popular do mercado de computação pessoal foi o *MS-DOS* da *Microsoft*. O sistema em questão era monousuário, monotarefa e gerenciava até 640 *kbytes* de memória base, características que para a grande maioria dos usuários poderia ser considerada satisfatória, mas que para profissionais e estudantes da área trazia uma realidade muito diferente da encontrada no grande porte, em especial nas universidades. No

início da década de 90, um estudante da Universidade de Helsinki na Finlândia, postava uma mensagem histórica na *Usenet* que revelava a programadores do mundo todo informações iniciais sobre o desenvolvimento de um *kernel* - de sua autoria, criado para fazer parte de um sistema operacional hipotético, construído a partir de suas experiências acadêmicas com o *minix* e que sobretudo, tinha a ambição de ser funcionalmente equivalente ao *Unix* clássico, porém voltado à arquitetura IBM PC. Para tanto, estava disposto a compartilhar o código fonte de seu trabalho com programadores de qualquer parte do mundo que eventualmente estivessem dispostos a trabalhar de maneira colaborativa no desenvolvimento e otimização do próprio *kernel*, bem como de toda a *userland* necessária para que o trabalho até então desenvolvido, passasse a constituir um sistema operacional de fato. A mensagem teve uma repercussão extremamente positiva e o projeto obteve a colaboração de programadores de diversas regiões do mundo, tornando-se um marco na história da computação: o estudante em questão era *Linus Torvalds* e o sistema operacional, então em fase inicial de desenvolvimento, o *Linux*. Desde então, os chamados sistemas operacionais modernos têm se apoiado em sistemas como o *Linux* e os *BSD's* para tirar o melhor proveito possível do *hardware* a que se destinam.

Vencida a barreira das limitações de *software*, a indústria dos semicondutores passou a ditar o ritmo da chamada "guerra dos *megahertz*" que impulsionou o desenvolvimento da computação pessoal na década de 90 e no início dos anos 2000. As máquinas a cada dia mais performáticas passaram então a ser utilizadas em grande escala e em diferentes segmentos, proporcionando diferenciais importantes no mercado a empresas que estivessem dispostas a investir na digitalização de seus processos.

2.3. A importância da virtualização

À medida que a adoção de computadores passou a ser uma realidade, não demorou muito tempo para que profissionais e pesquisadores da área percebessem que muitos dos recursos disponíveis nesse tipo de dispositivo permaneciam ociosos durante grande parte do tempo, com exceção de eventuais e esporádicos momentos

em que picos isolados de processamento porventura ocorressem. Em um sistema operacional multiusuário e multitarefa tradicional, normalmente os usuários precisam apenas de uma seção de trabalho aberta para realização de suas atividades que, em geral, consumem poucos recursos do equipamento. Ainda que vários usuários estejam conectados a um mesmo servidor, onerar os recursos de um equipamento dessa natureza de forma significativa, requer centenas ou milhares de operações simultâneas em grande parte dos serviços. Em termos práticos, tal realidade implica em grande desperdício de recursos de capital – ou simplesmente *CapEx*, frente ao investimento realizado pelas empresas para aquisição de seu parque computacional.

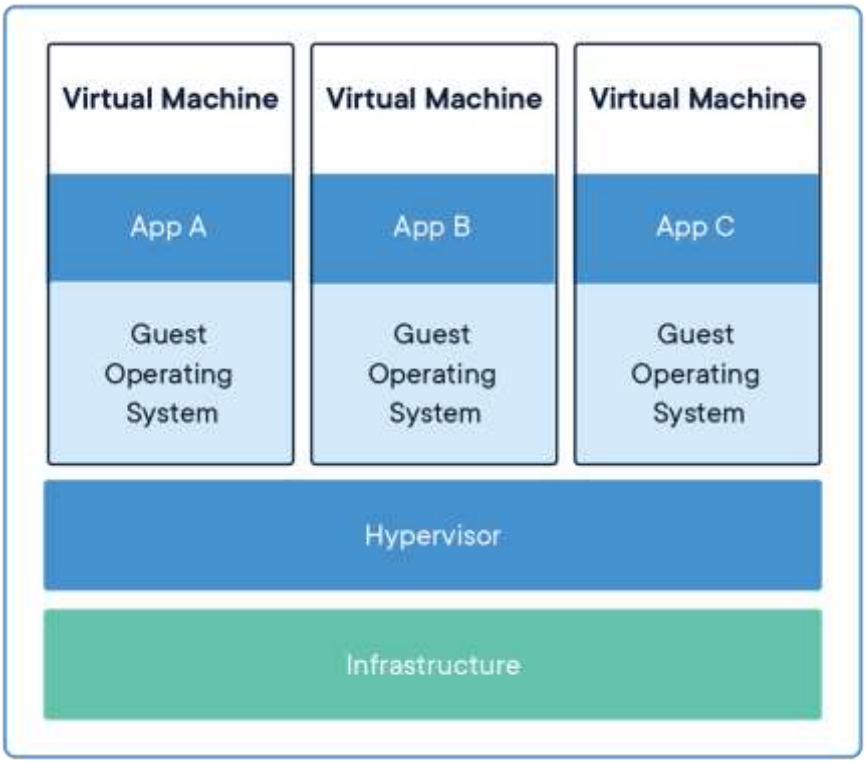
Em resposta à essa percepção, novas tecnologias passaram a ser idealizadas com o objetivo de possibilitar o particionamento dos recursos do sistema, possibilitando assim, a distribuição e o compartilhamento destes entre cargas de trabalho distintas, paralelas e logicamente isoladas, de modo a preservar a confidencialidade, integridade e disponibilidade dos dados em cada uma das instâncias em execução - nascia assim, a virtualização.

Segundo Docker (2021?), máquinas virtuais são uma abstração do *hardware* físico que permitem que um servidor possa ser fracionado em vários outros, na forma de instâncias virtualizadas. O *hypervisor* permite que várias máquinas virtuais sejam executadas em uma única máquina física.

Nesse sentido, tecnologias de virtualização passaram a figurar como uma alternativa factível para melhor aproveitamento dos recursos computacionais frequentemente ociosos. Apesar de continuar em uso até os dias atuais, a virtualização apresenta alguns pontos dignos de atenção, sobretudo quando empregadas em aplicações modernas que eventualmente façam uso de computação em nuvem, *pipelines* de *CD/CI* e demais práticas normalmente utilizadas na cultura *DevSecOps.* Dentre esses fatores, destacam-se o tempo de *bootstrap* do sistema operacional convidado, a necessidade de manutenção periódica das *appliances* frente a problemas de versionamento de *softwares* e bibliotecas de *runtime* — tal como em equipamentos físicos, bem como a dificuldade de reprodutibilidade de ambientes entre *pipelines* de desenvolvimento, testes, homologação e produção.

Cada máquina virtual inclui uma cópia completa de um sistema operacional, aplicativos, binários e demais bibliotecas necessárias — ocupando dezenas de gigabytes. A Figura 01 ilustra a arquitetura utilizada nas tecnologias de virtualização.

Figura 01 – Como funcionam as máquinas virtuais.



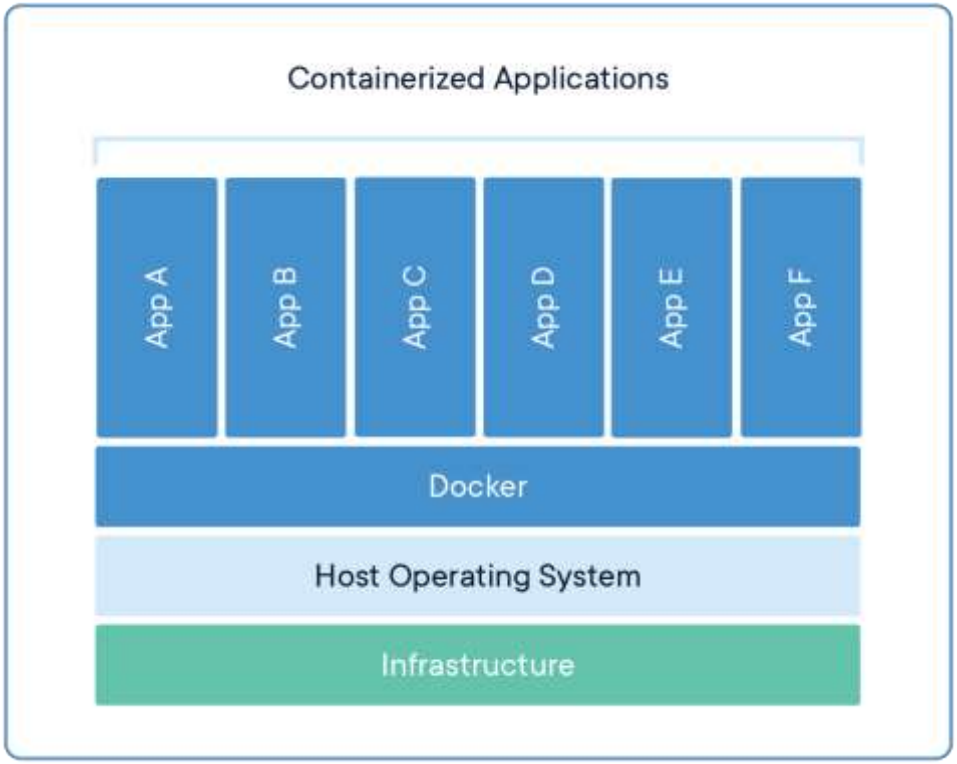
Fonte: Docker (2021?).

3. CONTAINERS

Segundo Docker (2021?), *containers* e máquinas virtuais tem benefícios semelhantes no que se refere ao isolamento e alocação de recursos, no entanto, funcionam de maneira diferente, tendo em vista que os *containers* virtualizam o sistema operacional e não o *hardware*, tornando-os mais portáteis e eficientes. Ao contrário das máquinas virtuais tradicionais, os *containers* utilizam o *kernel* do sistema operacional da máquina *host.* As instâncias são baseadas em imagens reutilizáveis que empacotam tudo o que é necessário para a execução da aplicação encapsulada na imagem em questão — em geral códigos-fonte, configurações, bibliotecas e

demais dependências, gerando como resultado um pacote de *software* leve e autônomo que pode ser executado de forma rápida e confiável em qualquer ambiente de computação que suporte a tecnologia. A Figura 02 ilustra a arquitetura utilizada nas tecnologias de virtualização baseadas em *containers*.

Figura 02 – Como funcionam os *containers*.



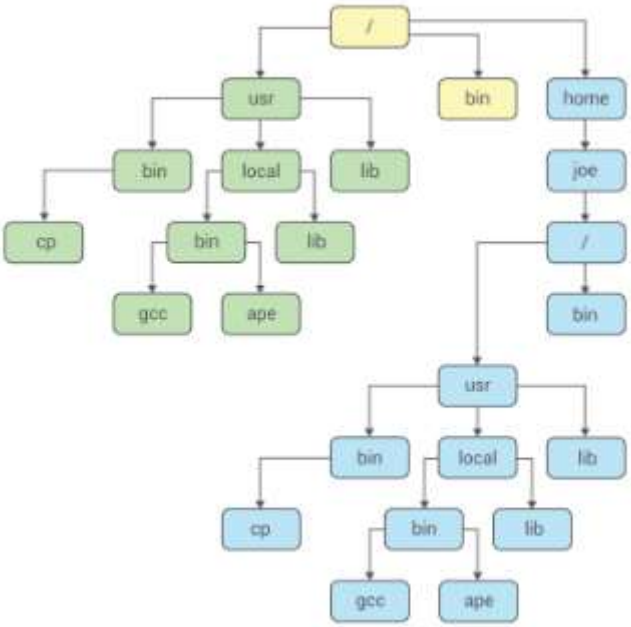
Fonte: Docker (2021?).

De acordo com Vitalino e Castro (2018), A utilização de mecanismos de virtualização baseados em *containers* não é algo recente. Desde o final da década de 70, diferentes tecnologias foram implementadas e aprimoradas de modo a possibilitar o particionamento de recursos do sistema em fluxos de processamento paralelos e isolados. As seções a seguir apresentam as principais.

3.1. Chroot

Segundo a Intel (2021?), ambientes *chroot* são parte fundamental no alicerce de tecnologias de virtualização “modernas” que, já há algum tempo, estão em grande evidência: os *containers*, por exemplo. O recurso introduzido na versão 7 do *Unix* clássico pela *Bell Laboratories* em 1979 e também no *4.2 BSD* da Universidade de Berkeley em 1983, consiste em um mecanismo de segurança capaz de redefinir a referência original do diretório raiz do sistema operacional para uma localidade diferente do sistema de arquivos, definida previamente e atrelada a determinado processo em execução, que em geral, representa um serviço em modo *listening*. Tal operação tem como objetivo teórico isolar o referido processo do sistema de arquivos principal, de modo a não comprometer a segurança de todo o sistema caso o binário seja alvo de algum ataque remoto, por exemplo. No entanto, apesar de sua relevância e propósito históricos, com o passar dos anos foi possível comprovar que ambientes *chroot* tradicionais eventualmente podem ser subvertidos de modo a comprometer o sistema principal. A Figura 03 demonstra um exemplo de abstração da raiz original do sistema operacional para uma versão alternativa disponível à partir do diretório */home/joe*.

Figura 03 – Exemplo de ambiente *chroot*.



Fonte: CodeProject (2021?) [3]

3.2. Jails

Dadas as fragilidades e limitações existentes nos ambientes *chroot*, novas tecnologias passaram a figurar como protagonistas em cenários onde a virtualização baseada em *containers* se fez presente – as *jails* são um exemplo. Inspiradas no conceito de *chroot*, surgiram pela primeira vez no sistema operacional *FreeBSD* 4.0 por volta dos anos 2000, permanecendo em uso até os dias atuais, com grande ênfase em segurança (KAMP *et al*, 2000). Seu grande diferencial consiste em implementar uma *userland* completa para cada *jail* em execução, desse modo, cada instância possui seu próprio sistema de arquivos, base de usuários, gerenciamento de pacotes e visibilidade de processos totalmente independentes do sistema principal, com exceção do *kernel*, que é compartilhado entre o sistema *host* e cada uma das *jails* em

[3] CodeProject. What is: chroot – The System Call and Utility in Linux. Disponível em: https://www.codeproject.com/Articles/1330321/What-is-chroot-The-System-Call-and-Utility-in-Linu. Acesso em: 20 fev 2021.

execução. Apesar de possuírem ferramentas de gerência amigáveis voltadas à produtividade, em geral são utilizadas de forma manual e estão disponíveis apenas em alguns sabores de *BSD*, o que de certo modo, limita sua popularidade, evolução e ampla adoção.

3.3. LXC

A tecnologia dos *Linux Containers* – ou simplesmente *LXC*, foi criada na série 2.6.26 do *kernel Linux* em 2008 e atua como alicerce de soluções mais recentes como o *Docker* - lançado em 2013, sendo apontada como grande responsável por seu sucesso.

De acordo com LINUXCONTAINERS (2021?), os *containers LXC* são frequentemente considerados como uma solução intermediária entre um ambiente *chroot* clássico e uma máquina virtual completa. No entanto, uma definição mais assertiva os definem como um ambiente o mais próximo possível de uma distribuição *Linux* padrão, sem a necessidade de um *kernel* dedicado.

Em termos práticos, a tecnologia implementa recursos fundamentais para o isolamento de processos e dos recursos de rede, memória, disco e *CPU* entre diferentes *containers* e o *host* hospedeiro.

3.4. Docker

O *Docker* pode ser considerado como uma suite de ferramentas voltada à construção, compartilhamento e execução de aplicações utilizando a tecnologia de *containers*.

De acordo com Docker (2021?), é responsável por eliminar as tarefas de configuração rotineiras e repetitivas, sendo utilizado em todo o ciclo de vida de desenvolvimento, de modo a possibilitar o desenvolvimento de aplicativos portáveis de modo rápido, fácil em ambientes *desktop*, *web* e *cloud*. A plataforma inclui *API's* e segurança projetadas para trabalhar juntas em todo o ciclo de vida de entrega do aplicativo.

4. RESULTADOS

Com o objetivo de demonstrar os principais recursos da tecnologia de *containers*, foi utilizada a plataforma "*Play with Docker*", definida por seus criadores como um "parque de diversões" divertido, simples e interativo voltado para o aprendizado da ferramenta *Docker*, cuja página principal pode ser visualizada na Figura 04.

Figura 04 – Plataforma de estudo em nuvem: *Play with Docker.*

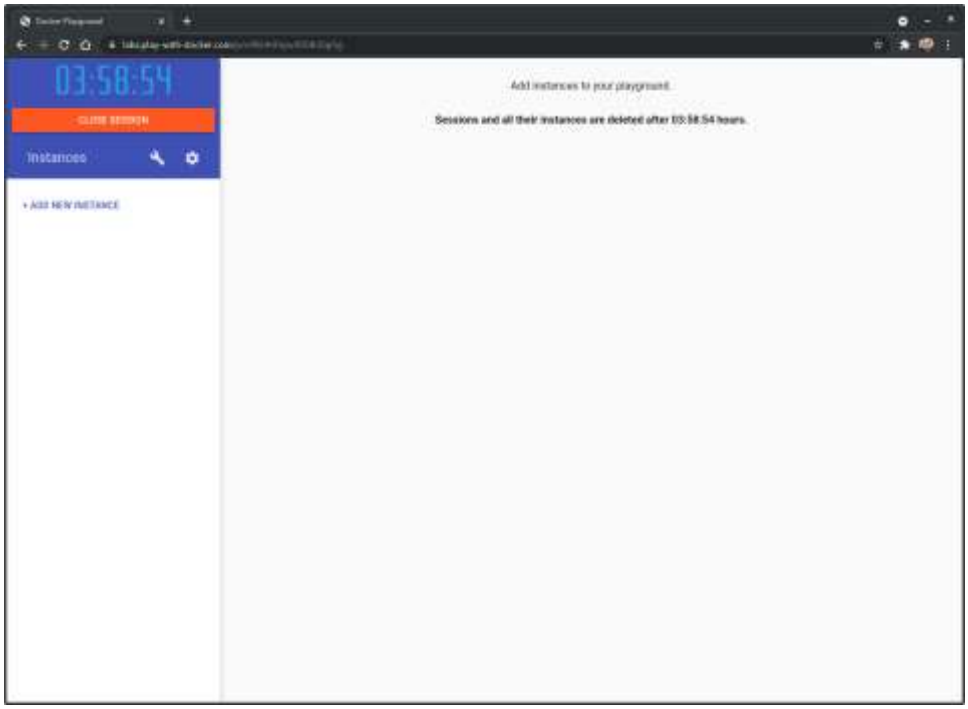
Fonte: Labs.play (2020)[4]

A utilização da plataforma é gratuita, a única limitação imposta pelos desenvolvedores é a restrição do tempo de provisionamento das instâncias, condicionado ao período máximo de 4 horas, conforme mostra a Figura 05. Após esse período, o ambiente é encerrado e as informações locais perdidas. Caso o usuário tenha a necessidade de continuar a utilizar o ambiente após o período supracitado,

[4] Labs.play. Play with Docker - a simple, interactive and fun playground to learn Docker. Disponível em: https://labs.play-with-docker.com/. Acesso em: 20 mar 2021.

basta que inicie um novo ambiente que, de mesmo modo, poderá ser utilizado por igual período. Como a necessidade dos autores é somente demonstrar os principais recursos disponíveis nas tecnologias de *containers*, o ambiente é plenamente suficiente.

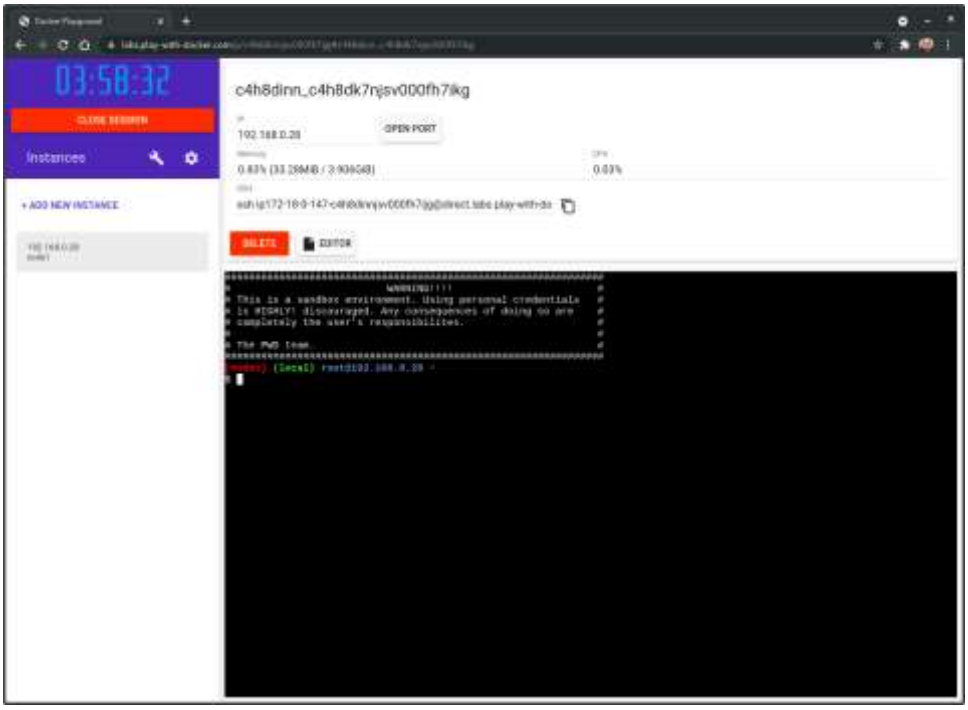
Figura 05 – *Play with Docker*: Tela inicial.



Fonte: Autoria própria.

Na Figura 06, é possível observar no menu do lado esquerdo ao topo, o timer referente à sessão atual e logo abaixo o botão "+ *ADD NEW INSTANCE*" que, quando pressionado, disponibiliza uma "*Docker Machine*" totalmente funcional - denominada "*node1*", que pode ser utilizada clicando ao lado direito da tela.

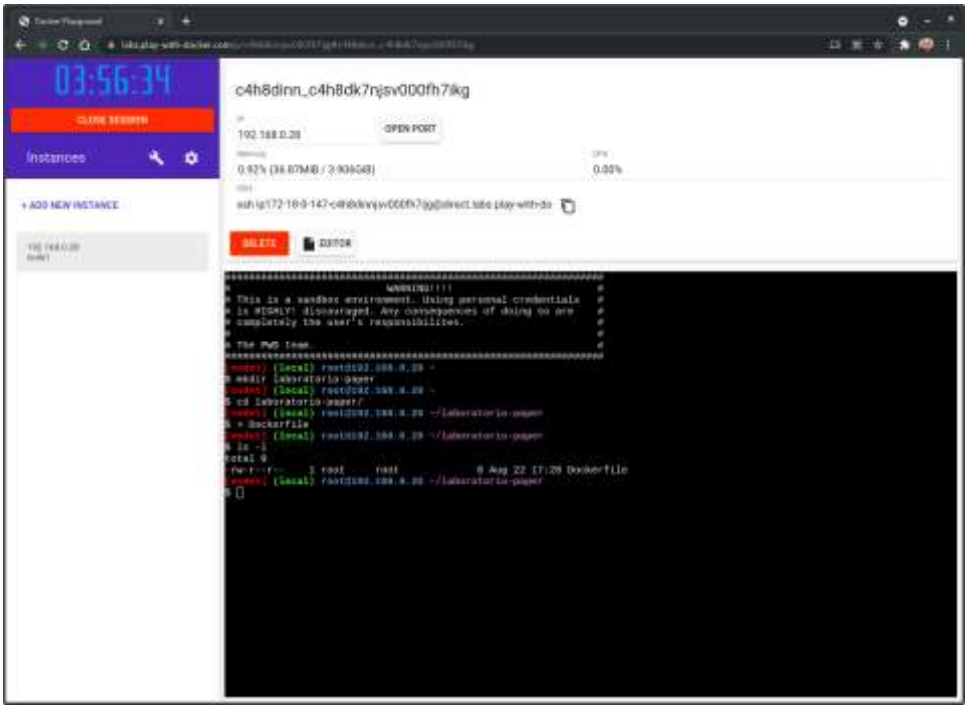
Figura 06 – *Play with Docker*: Criação de um nó de trabalho.



Fonte: Autoria própria.

Conforme descrito nas seções anteriores, uma das principais características da tecnologia de *containers* é possibilitar a criação de imagens reutilizáveis. Para tanto, as instruções desejadas devem ser incorporadas a um arquivo específico, denominado *Dockerfile*. A Figura 07 traz os passos necessários para a preparação do ambiente e a criação do arquivo supracitado.

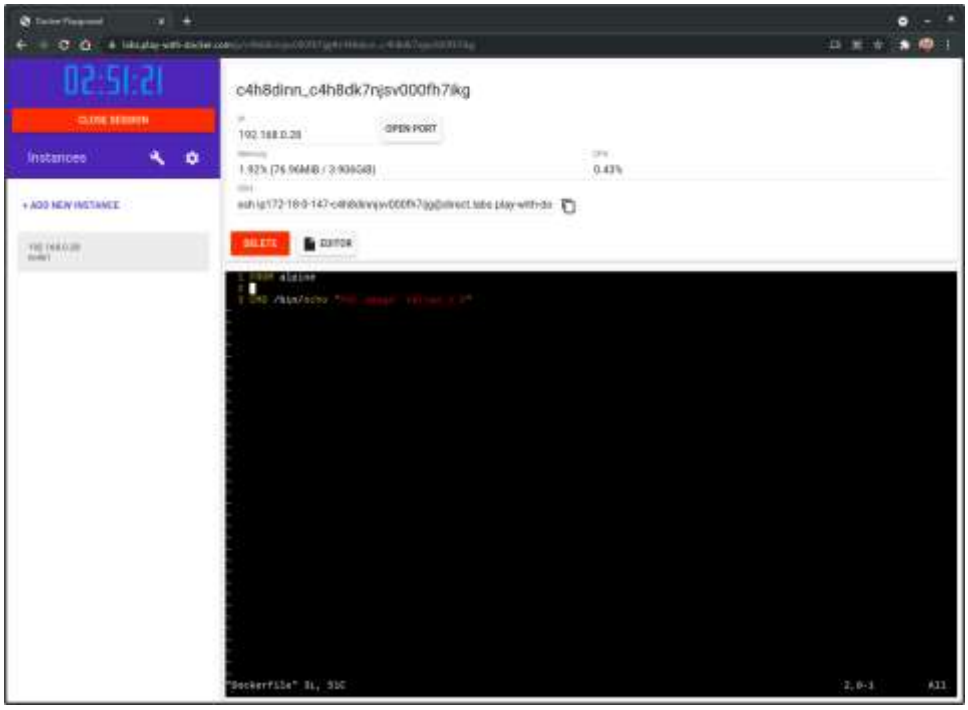
Figura 07 – *Play with Docker*: Criação do primeiro Dockerfile.



Fonte: Autoria própria.

A título de prova de conceito – *PoC*, foi criado um *Dockerfile* composto de duas instruções que respectivamente fazem uso da imagem do *Alpine Linux* e executam um comando que imprime na tela a mensagem “*PoC image*: Versao 1.0”. A criação do *Dockerfile* pode ser visualizada na Figura 08.

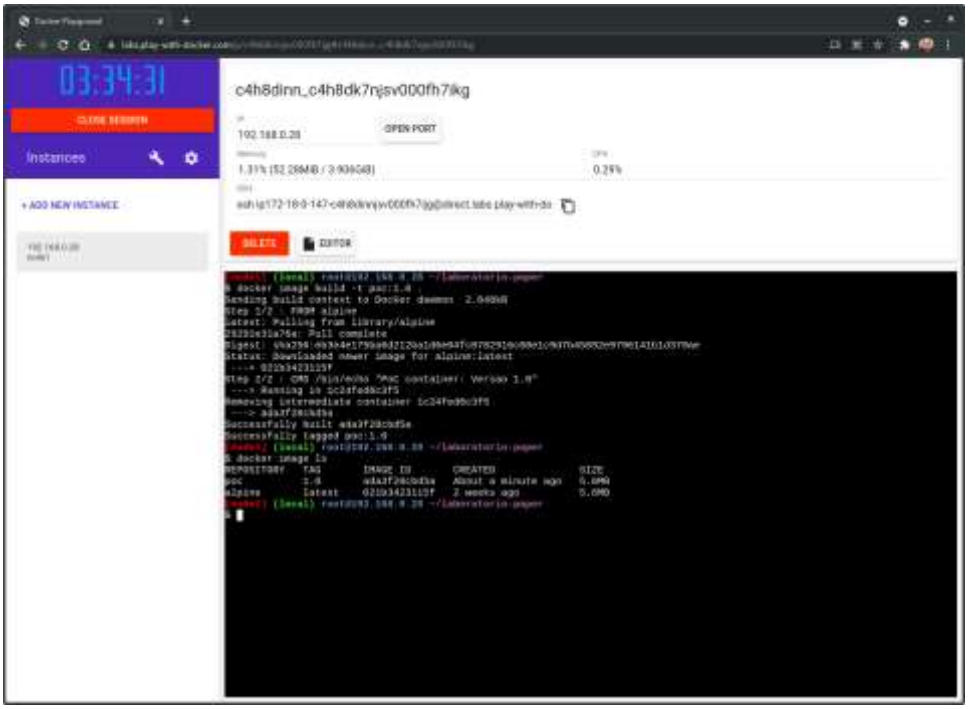
Figura 08 – *Play with Docker*: Código fonte da imagem.



Fonte: Autoria própria.

Para que o *Dockerfile* possa se tornar efetivamente uma imagem e posteriormente servir como *template* para um ou mais *containers*, é preciso que seja realizado o processo de *build* da imagem. No exemplo em tela, a imagem construída receberá o título de "*poc*" e será versionada com o identificador 1.0, conforme demonstrado na Figura 09.

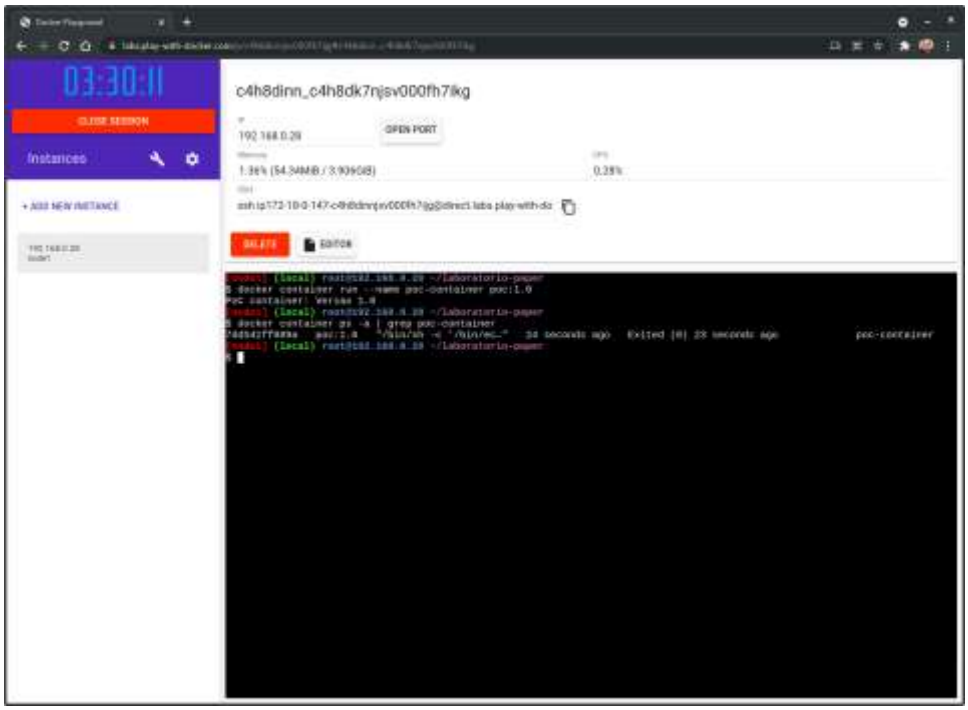
Figura 09 – *Play with Docker*: Processo de "*build*" da imagem.



Fonte: Autoria própria.

Uma vez construída a imagem, esta ficará imediatamente disponível localmente, podendo ser utilizada quantas vezes forem necessárias e bastando para tanto que *containers* que a (re)utilizem sejam instanciados. A Figura 10 ilustra o processo de criação de um container denominado "*poc-container*" que utiliza a imagem "*poc:1.0*" criada nos passos anteriores e que, conforme esperado, imprime a mensagem "*PoC image*: Versao 1.0" no *console*.

Figura 10 – *Play with Docker*: Execução de um container baseado na imagem “poc” na versão 1.0.



Fonte: Autoria própria.

5. DISCUSSÕES E CONCLUSÕES

A tecnologia de *containers* possibilita de maneira simples, eficiente e confiável a reprodução de ambientes completos em *pipelines* distintas de desenvolvimento, testes, homologação e produção, independentemente do sistema operacional utilizado (*Linux, MacOS ou Windows*) ou do modelo de infraestrutura adotado (*on premises, cloud* ou híbrido). Para melhor aproveitamento da tecnologia recomenda-se que as aplicações que façam uso do paradigma tenham a arquitetura projetada para o modelo de micro serviços e que os profissionais envolvidos estejam inseridos na cultura *DevSecOps*.

O presente trabalho demonstrou, a título de prova de conceito, como construir uma imagem minimalista - baseada na distribuição *Alpine Linux,* que rapidamente pode ser reutilizada por um ou mais *containers*. A imagem construída ocupa apenas 5.6 megabytes de armazenamento e sua disponibilidade foi restrita somente à

plataforma utilizada para a *PoC*. Uma aplicação que porventura tivesse a necessidade de ser distribuída a um número maior de usuários poderia ter esse requisito facilmente atendido em poucos segundos, via *upload* da imagem para um *registry* privado disponível na rede local ou ainda, para repositórios na nuvem como o *Docker Hub* ou o *Amazon Elastic Container Registry (ECR),* com possibilidade de disponibilidade pública ou restrita a um público privado determinado previamente.

Na demonstração apresentada, foi utilizada somente uma "*Docker Machine*", representada pelo *host* denominado *node1*. Em ambientes de produção, envolvendo aplicações de missão crítica, recursos de alta disponibilidade e elasticidade são tidos como essenciais. A tecnologia de *containers* oferece soluções de orquestração robustas que atendem a esse tipo de propósito, como o *Docker Swarm* e o *Kubernetes*.

De mesmo modo, é possível ainda a construção de topologias complexas envolvendo serviços, redes e volumes de armazenamento utilizando soluções de infraestrutura como código como o *Docker Compose*, de modo a facilitar a eventual transição da infraestrutura *on premises* para o paradigma de computação em nuvem ou mesmo para uma implementação híbrida.

Os recursos supracitados serão objeto de pesquisa em trabalhos futuros, podendo também ser explorados pelo leitor após consulta à documentação das ferramentas.

REFERÊNCIAS BIBLIOGRÁFICAS

DOCKER. **Use containers to Build, Share and Run your applications.** 2021?. Disponível em: https://www.docker.com/resources/what-container. Acesso em: 22 ago 2021.

INTEL. **Linux Containers Streamline Virtualization and Complement Hypervisor-Based Virtual Machines.** 2021?. Disponível em: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/linux-containers-hypervisor-based-vms-paper.pdf. Acesso em: 10 jan 2021.

LINUXCONTAINERS. **What's LXC?.** 2021? Disponível em: https://linuxcontainers.org/lxc/introduction/. Acesso em: 22 ago 2021.

KAMP et al. **Jails: Confining the omnipotent root.** InSANE, 2000. 116 p.

VITALINO, J. F. N.; Castro, M. A. N. **Descomplicando o Docker.** 2a Edição. Editora Brasport, 2018.

Silva, W. F. da. **Aprendendo Docker – Do Básico à Orquestração de Contêiners.** Editora Novatec, 2016.