Nathan Batista de Oliveira

# Aplicações em Contêineres

São José - SC

Março/2017

Nathan Batista de Oliveira

# Aplicações em Contêineres

Monografia apresentada à Coordenação do Curso Superior de Tecnologia de Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do título de Tecnólogo em Sistemas de Telecomunicações.

Instituto Federal de Santa Catarina – IFSC

Campus São José

Sistemas de Telecomunicações

Orientador: Ederson Torresini

São José - SC

Março/2017

Nathan Batista de Oliveira
Aplicações em Contêineres/ Nathan Batista de Oliveira. - São José - SC, Março/2017
Orientador: Ederson Torresini

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Sistemas de Telecomunicações, Março/2017.

1. Contêiner. 2. Docker. 2. Kubernetes. 3. CoreOS. I. Ederson Torresini. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Disposição de Aplicações em Contêineres

Nathan Batista de Oliveira

# Aplicações em Contêineres

Monografia apresentada à Coordenação do Curso Superior de Tecnologia de Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do título de Tecnólogo em Sistemas de Telecomunicações.

Trabalho aprovado. São José - SC, 17 de março de 2016:

---

**Ederson Torresini**
Prof. Me.

---

**Odilson Tadeu Valle**
Prof. Dr.

---

**Humberto José de Sousa**
Analista de TI

São José - SC
Março/2017

# Agradecimentos

Ao Prof. Orientador Ederson Torresini, por toda a presteza, paciência e disponibilidade em contribuir com a construção desta monografia, braço amigo de todas as etapas deste trabalho.

À minha família, pela confiança, motivação e todo o suporte possível para que eu tivesse condições de concluir este curso.

Aos professores, amigos e colegas de curso pela ilustre companhia, pela força, pelo incentivo e por todo e qualquer auxílio que tenha contribuído a trilhar esta jornada.

Aos demais profissionais do campus pela disponibilidade e suporte aos alunos sempre que necessário.

A todos que, com boa intenção, colaboraram para a realização e finalização deste trabalho.

Ao professor coordenador do curso Alexandre Moreira, por toda paciência e boa vontade ao ceder-me o tempo que foi preciso para o término desta monografia.

*"Um mundo no qual o tempo é absoluto é um mundo consolador.*
*Pois, embora os movimentos das pessoas sejam imprevisíveis, o movimento do tempo é previsível.*
*Embora se possa duvidar das pessoas, não se pode duvidar do tempo.*
*Enquanto as pessoas ficam divagando, o tempo prossegue em sua caminhada sem olhar para trás."*
*(Alan Lightman)*

# Resumo

Esta monografia parte do ponto onde tem-se a infraestrutura de aplicações e serviços do câmpus São José do Instituto Federal de Santa Catarina (IFSC), no qual tem-se máquinas virtuais dedicadas com seus próprios recursos computacionais. Com o projeto, pretende-se implantar o mesmo ambiente, lidando com sobrecargas de processamento dos serviços, bem como eventuais falhas que possam ocorrer com a aplicação durante sua utilização, através da abordagem de Contêiner como um serviço (*CaaS*), cujo objetivo é uma nova infraestrutura mais leve, rápida, facilmente escalável, com bom nível de abstração para implantação de aplicações. Este método consiste em um isolamento de serviços em nível de processo, permitindo uma infraestrutura de serviços flexível e de tamanho variável, onde cada unidade de virtualização sobre contêiner pode ser recriada, duplicada ou removida de acordo com a demanda de acesso nas aplicações contidas nestes contêineres, promovendo economia de recursos de *hardware*, *software* e rede.

# Abstract

This monograph starts from the point where the infrastructure of applications and services of São José Campus of the Federal Institute of Santa Catarina (IFSC), in which there are virtual machines dedicated with their own computational resources. With the project, it is intended to implement the same environment, dealing with service processing overloads, as well as any failures that may occur with the application during its use, through the Container approach as a service (CaaS) Whose goal is a new, lightweight, fast, easily scalable infrastructure with a good level of abstraction for application deployment. This method consists of process-level service isolation, allowing for a flexible and variable-size service infrastructure where each container-based virtualization unit can be recreated, duplicated or removed according to the demand for access in the applications contained in these containers , saving hardware, software and network resources.

**Keywords**: Contêiner. CaaS. SaaS. Orquestração.

# Sumário

# Lista de ilustrações

# 1 Introdução

Quando se fala em *sites*, *webservices* e outras aplicações, é impossível não mencionar o quão crescente é o acesso de usuários em sistemas de lojas, prestadoras de serviços, empresas terceirizadas, entre outros, em ambientes hospedados em servidores Web; ou seja, a importância de um *webservice* para a grande maioria das instituições. Um dado que justifica essa afirmação é uma pesquisa realizada pela *NetCraft*, página especializada estatísticas com relação ao uso de servidores Web, onde claramente esse aumento é visto através da Figura 1. Neste caso, foi realizado um mapeamento da década de 2000, verificando o total de servidores executando os determinados serviços e ambientes, conforme legenda (Netcraft Ltd., 2010).

O que fica evidente é o fato de que por trás de todas essas páginas, textos, imagens, serviços, entre outros, há sempre uma infraestrutura física e virtual que mantém disponível esses ambientes para acesso e visualização por parte dos usuários, seja uma loja virtual, uma página de pesquisa, uma documentação ou qualquer outro tipo de site ou serviço web. Um dos muitos cenários possíveis para desenvolvimento de aplicações ou serviços, comumente utilizado para sustentar esses sites, baseia-se em servidores de aplicação Web, que seguem uma ordem de desenvolvimento e versionamento, disponibilização através de um serviço Web e configuração de banco de dados, entre outras atividades (MORAES, 2015).

Tendo em questão essas prerrogativas, uma determinada aplicação ou *webservice* pode ser acessado através do navegador pelo usuário. Uma falha em qualquer ponto da aplicação implica, pois, uma necessidade de correção e, portanto, indisponibilidade. Outra grande adversidade que é encontrada em alguns momentos, é o fato dos serviços estarem sujeitos ao desempenho dos servidores virtuais ou físicos que os hospedam, seja esse um desempenho satisfatório ou precário.

Têm-se, atualmente, um grande número de páginas de muitas instituições ou empresas com base em virtualização de servidores, disponibilizados para tal demanda, conforme acontece com as aplicações do ambiente virtual do campus São José do Instituto Federal de Santa Catarina.

Atualmente, no câmpus os serviços Web operam sobre uma infraestrutura virtualizada. Embora seja um método bastante comum por contar com relativa confiabilidade, possibilidade de criação facilitada de ambientes, bom nível de segurança e a possibilidade de criação de ambientes para os mais variados serviços, a estrutura atual conta com o fato de ter muitos recursos que poderiam ser melhor aproveitados utilizando de uma construção mais leve que tem por objetivo a economia de recursos computacionais no ambiente.

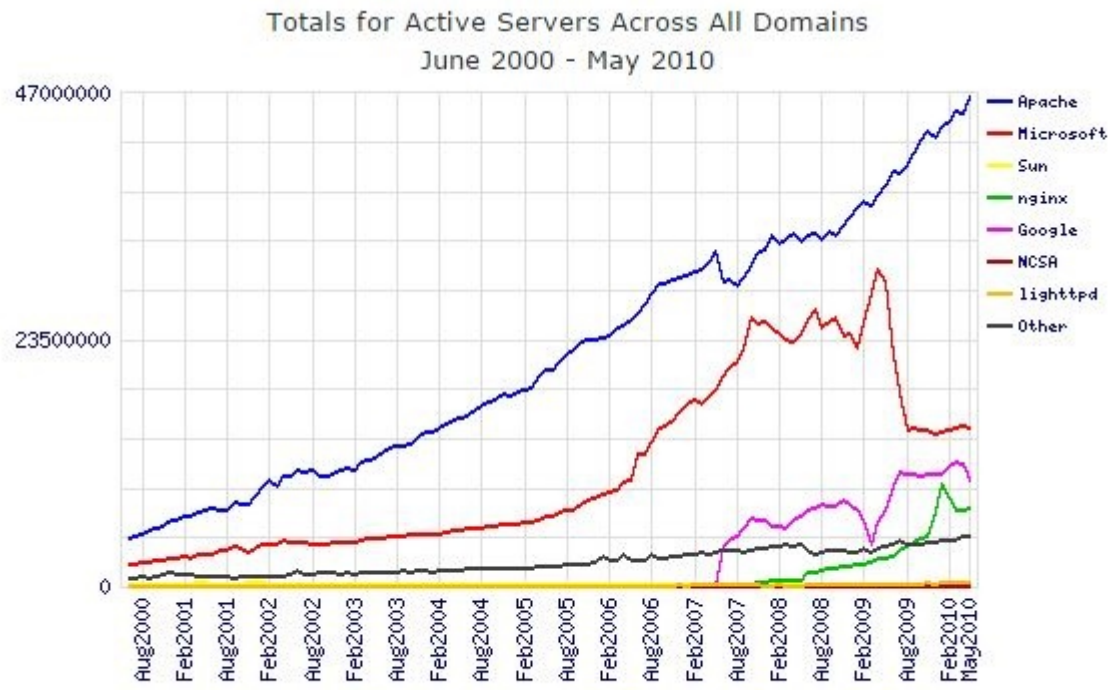


Figura 1 – Servidores Web mais utilizados. Fonte: NetCraft Ltd.

Como um possível solução para o cenário exposto, pode-se destacar uma solução baseada no conceito de contêineres. Em linhas gerais, trata-se de uma solução que consiste em uma virtualização em nível de sistema operacional - isolamento de recursos como rede, processo, sistema de arquivos e outros. Para uma melhor compreensão do conceito, pode-se pensar que cada contêiner em um *host* é um processo isolado e independente, contendo um determinado serviço em seu escopo representado por uma imagem deste serviço, seja ele uma aplicação, um *webservice*, um banco de dados, entre outros. Tal solução permite redução de recursos de servidores para manutenção e monitoramento, mobilidade de serviços entre diferentes sistemas operacionais de distribuição Linux, possibilidade de execução mais de uma instância de um serviço no mesmo servidor, capacidade de escalar conforme variação de demanda além de melhoria de desempenho provida por uma redução de sobrecarga, vantagens estas que serão exploradas adiante.

## 1.1 Objetivos

Assume-se uma infraestrutura mais simples algo que conte com o fato de que os elementos ou ferramentas que a componham sejam mais leves, ocupem menos espaço e trabalhem somente de acordo com demanda de serviço. Têm-se, então, como objetivo, implantar uma infraestrutura baseada no método de contêineres, tendo elementos como as unidades de isolamento de contêineres, um sistema operacional mais leve e preparado para

trabalhar com esta solução e uma ferramenta que gerencie estes contêineres de acordo com a demanda recebida.

A infraestrutura física disponibilizada para as aplicações e serviços para a realização do experimento conta com três estações de trabalho, contando com processadores Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz, ambas com 2GB de memória RAM e 32GB de armazenamento, o que considera-se uma configuração relativamente pequena.

Contando com tal infraestrutura física, pretende-se criar um determinado número de máquinas virtuais com sistema operacional voltado para aplicações com contêineres em execução ou então utilizar diretamente as máquinas físicas. Nessas máquinas, o sistema operacional que será instalado deve responsabilizar-se por atividades como início de determinados serviços, especificação de algumas métricas e principalmente a comunicação entre essas várias máquinas virtuais caracterizando, assim, o funcionamento em *cluster* das máquinas.

Propõe-se, pois, a instalação de uma ferramenta responsável pelo gerenciamento e orquestração dos contêineres, que deve automatizar o redimensionamento da infraestrutura, sendo necessárias funcionalidades como controle de replicação de unidades de contêiner, endereçamento de contêineres, mapeamento dos conteúdos, entre outras características. A orquestração também pode encarregar-se de atividades como monitoramento, balanceamento de carga e resolução de nomes de domínio.

Ainda que mencionados o tipo de sistema operacional mais compatível com contêineres, o tipo de contêiner que será utilizado bem como a ferramenta de gerenciamento e orquestração, não será definido em princípio um número fixo de contêineres pelo fato de tratar-se de uma infraestrutura flexível. Mesmo assim, presume-se que sejam pré-definidos valores máximos e mínimos de contêineres, de acordo com o tipo de aplicação.

Em linhas gerais, pensa-se em uma virtualização em nível de processo, na qual os serviços são encapsulados, onde essas unidades de virtualização sejam hospedadas em um sistema operacional propício para o cenário, e contém com uma ferramenta que gerencie a inicialização, pausa ou parada dos serviços encapsulados de acordo com a demanda.

# 2 Fundamentação Teórica

Alguns obstáculos podem surgir nos métodos atuais de implantação, conforme anteriormente citados e, tendo em mente a necessidade de inovar, têm-se buscado uma solução para a camada de infraestrutura de aplicações de um determinado ambiente baseado no conceito de contêineres *web*. Esse método de hospedagem de aplicações pode ser visto também como uma nova forma de se enxergar o ambiente em relação às implantações desses aplicativos, como um novo paradigma.

Apesar de ter feito sucesso de forma muito recente e ter tornado-se uma opção muito utilizada para a plataforma de aplicações há pouco tempo, é válido lembrar que as primeiras implementações baseadas no conceito de contêineres foram feitas na metade da década de 2000, em particular Verma et al. (2015), o qual trouxe o conceito de *cgroups* no *Kernel* Linux. Os *cgroups* são os grupos de controle que podem ser definidos como uma característica nativa do *Kernel* podendo limitar, alocar, controlar, priorizar recursos de processamento, memória, vazão, para grupos de processos hierarquicamente organizados através de um controlador de recursos, usando para tal *namespaces* para garantir o isolamento entre os mesmos.

Além dos *cgroups*, há outras implementações de isolamento de recursos em outras variantes do UNIX. O *FreeBSD*, por exemplo, utiliza também um tipo de virtualização em nível de sistema operacional chamado *jails* que permite criar microssistemas a partir do princípio de modificação do diretório raiz, ou *chroot*, dando a chance de administradores implantarem seus sistemas de forma isolada sem comprometer o sistema.

Outras soluções foram lançadas posteriormente para melhorar o funcionamento do *FreeBSD jails* como *Solaris Zones*, que consiste em um ambiente virtualizado dentro de uma instância do *Solaris OS* onde, dentro de cada uma dessas zonas, é executada determinada aplicação de forma isolada e segura, a medida que os recursos do sistema operacional gerenciados de forma centralizada.

Atualmente, o isolamento de um determinado processo em *namespaces* distintos da-se o nome de contêiner. Comparando o modelo de contêineres com o modelo de máquinas virtuais, é possível perceber, conforme exemplo explícito através da Figura 2, que uma máquina que executa contêineres não utiliza um hipervisor (*hypervisor*), consequentemente, não contando com sistemas operacionais sobre máquinas virtuais, tendo somente a aplicação (*engine*) responsável pela criação das unidades de contêiner, neste caso a implementação de contêineres *Docker*.

Uma das principais vantagens de se utilizar a plataforma baseada em contêineres é a fácil escalabilidade desses aplicativos em comparação como uma mesma aplicação rodando

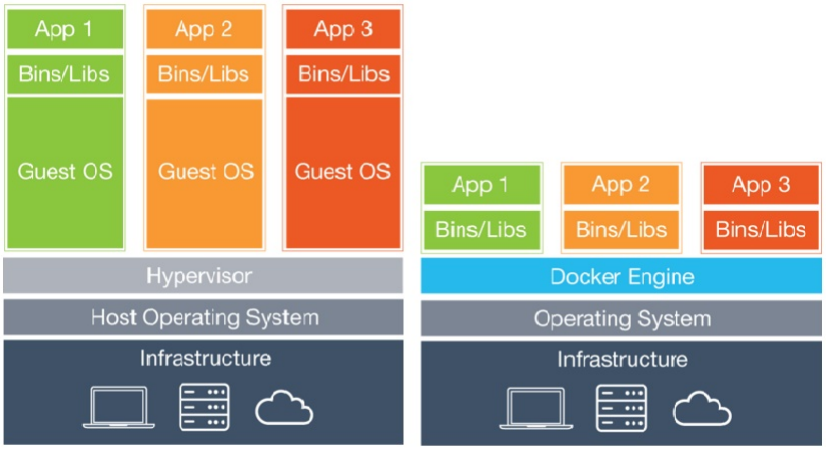


Figura 2 – Virtual Machine vs Container. Docker, Inc.

em máquina virtual isolada, uma vez que implica abstrair o *hardware* para esse último e, portanto, levar a perda de desempenho global de processamento. Outro fator interessante é a transparência, por parte de um contêiner, dentro dessa rede de aplicações. Tal fator deve-se à premissa de que, do ponto de vista da plataforma, todo contêiner é uma espécie de caixa com determinada aplicação dentro, sem importar-se com quaisquer características dessa aplicação. De forma recíproca um contêiner com determinado aplicativo também não preocupa-se com detalhes a respeito dos recursos da máquina que o hospeda.

Conforme citado no texto, contêineres permitem uma estrutura de funcionamento extremamente leve e rápida por não contarem com um hipervisor ou *Kernel* próprio, sendo executados como um processo e compartilhando com o *Kernel* do servidor os arquivos de bibliotecas, binários, arquivos de configurações, entre outros. Essa característica permite que inúmeros contêineres sejam executados a partir de uma determinada estação de trabalho. Vale ressaltar que pode ser utilizado também um determinado tipo de contêiner que contenha, em suas funcionalidades, uma opção de agregação (*clustering*), possibilitando a implantação de um grupo de contêineres, seja através do compartilhamento de recursos intra-isolamento ou mesmo por serviços auxiliares como autodescoberta de serviços em rede, entre outros.

No âmbito do gerenciamento desses contêineres, a ferramenta de administração e orquestração define, através de determinadas automatizações, as características e configurações dos contêineres onde ficam as aplicações. Tal propriedade concede a replicação de contêineres com aplicações já existentes no ambiente caso seja necessária uma substituição imediata de contêineres, seja por falhas ou necessidade de expansão destes para atender maior demanda.

Além do método de aplicações através dos contêineres e da utilização de uma ferramenta para gerenciamento e escalabilidade dos mesmos, faz-se necessária a utilização de um sistema operacional estável na infraestrutura, de forma que esse sistema tenha

características que se encaixem na solução de contêineres e gerenciamento, como uma maior tolerância a falhas, alto nível de abstração, bom comportamento ao funcionar em *cluster*, facilmente escalável, entre outras vantagens.

Consideram-se desvantagens da solução o fato de apresentar aumento de entrada e saída de dados em rede, levando em conta que também podem ser necessárias alterações na escrita das aplicações para adequá-las ao método. Por compartilhar recursos com o *Kernel* do servidor, também assume-se certa vulnerabilidade de segurança de informações. Analisando um servidor dentro de um *cluster* com suas unidades de contêiner em execução pode-se afirmar que, caso este servidor passe por problemas e apresente indisponibilidade, todos os contêineres que nele estão também são perdidos. Tal problema pode ser contornado com a implementação de vários servidores em *cluster* que podem suprir a falta das unidades de contêineres perdidas, criando novas unidades automaticamente com o gerenciador de contêineres, para assumir a demanda.

## 2.1 Sistema Operacional

Para apresentar os elementos pontuais de uma nova infraestrutura baseada em contêineres, começa-se a partir do que pode ser considerado o primeiro nível da topologia, o sistema operacional. A solução de contêineres foi pensada para a execução nas mais variadas distribuições *Linux*, como *Debian*, *Ubuntu* e *CentOS*, por exemplo.

Na busca por um sistema operacional com fácil escalabilidade para as aplicações distribuídas, bom grau de abstração e que suporte de forma plena a execução dessas aplicações em modo *cluster*, uma opção bastante interessante é o *CoreOS*. Pode-se afirmar que o *CoreOS* é recomendado pois depende de unidades de contêiner para administrar a implantação de serviços ou *webapps* no sistema e foi desenvolvido justamente com a finalidade de funcionar em *cluster*. Esse fato comprova-se por ele possuir poucos binários e sem quaisquer recursos de instalação e atualização de pacotes. Basicamente conta com seu *systemd* para gerenciamento e inicialização de seu *Kernel* e dos demais processos de maneira centralizada (CoreOS, Inc., 2015).

Independente de utilizar *CoreOS* ou qualquer outra distribuição, algumas ferramentas auxiliares são necessárias para prover características como o funcionamento em *cluster* e endereçamento das unidades de contêiner. De forma nativa, o *CoreOS* já possui, em suas especificações, um *runtime* para contêineres chamado *Rocket*, que trata-se de uma interface de linhas de comando (CLI) para inicialização e execução de unidades de contêiner que interage com o *systemd* e com os sistemas de orquestração de *cluster*, além da implementação de terceiros Docker (CoreOS, Inc., 2016).

De qualquer maneira uma infraestrutura que hospede este modelo pode funcionar muito bem com qualquer uma das distribuições *Linux*, ficando a critério do administrador

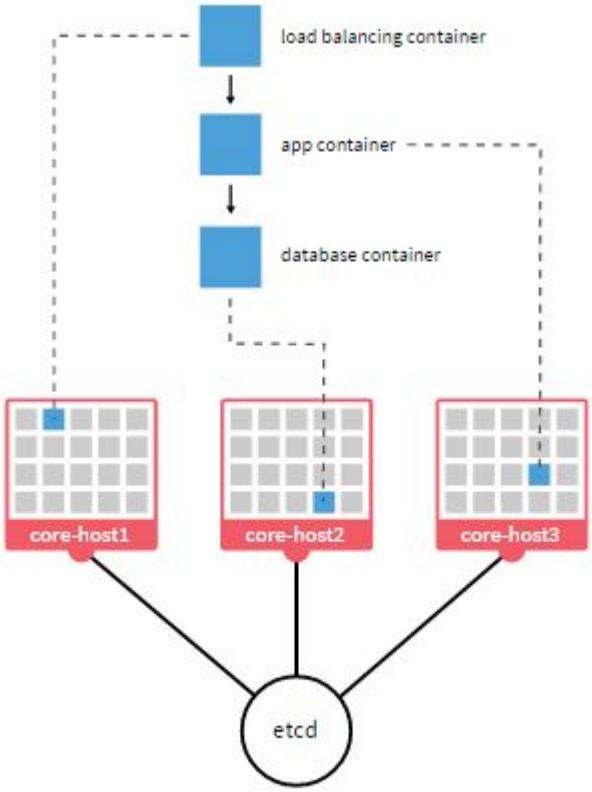


Figura 3 – Exemplo de aplicação Web usando contêineres. Fonte: CoreOS, Inc.

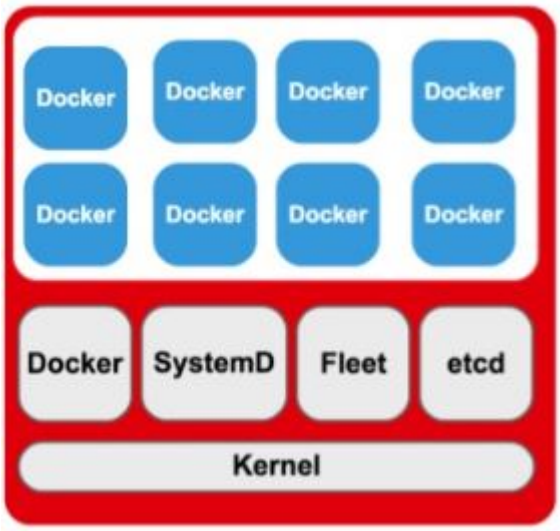


Figura 4 – Servidor com *CoreOS*

qual distribuição utilizar.

## 2.2 O Contêiner

Levando em consideração uma infraestrutura de máquinas já disponibilizadas com determinada distribuição e pronto para o funcionamento em cluster, parte-se então para o nível acima do sistema operacional, que trata-se do tipo de contêiner que será utilizado na solução.

Para este trabalho, será adotada a implementação de contêiner (Docker, 2015) por já ter suporte às outras ferramentas utilizadas no modelo, como sistema operacional *CoreOS* e melhor compatibilidade com a ferramenta de orquestração de contêineres.

A implantação de contêineres *Docker* torna-se possível por conta de uma espécie de console denominada *Docker Engine*, que é um forma de empacotamento de infraestrutura simplificada. A solução *Docker* utiliza, em suas especificações, recipientes de contêiner trabalhando no mesmo sistema operacional do servidor e usando um sistema de arquivos em camadas. Uma das grandes vantagens do *Docker* em si é que podem ser criados *snapshot* de determinada funcionalidade em uma imagem, simplificando a implantação da mesma em outros contêineres *Docker*, permitindo assim a replicação de estados de contêineres em escala.

## 2.3 Orquestração e Gerenciamento

Com o *cluster* dos servidores já configurado compartilhando informações sobre os contêineres e sub-redes designadas às unidades, faz-se necessária a implementação de uma ferramenta que gerencie todas essas unidades de contêiner, trabalhando com a possibilidade de eventuais falhas no ambiente e a necessidade de redimensionar a infraestrutura, essa ferramenta deve ser capaz de lidar com essas necessidades de forma plena e confiável. Neste caso, adota-se a ferramenta de administração e orquestração chamada *Kubernetes* (The Linux Foundation, 2016). É importante mencionar que essa abordagem de gerenciamento de contêineres iniciou-se através do gerenciador *Borg* (VERMA et al., 2015). O *Borg* é uma ferramenta capaz de gerir uma grande quantidade *clusters* de aplicações, permitindo a implantação de serviços com alta disponibilidade, garantindo encapsulamento de suas unidades, isolamento em nível de processo, integração de serviços, monitoramento e ferramentas para analisar o funcionamento do sistema. Baseado nesses princípios pelas facilidades que o *Borg* apresenta, foi lançado o orquestrador de contêineres *Kubernetes* justamente com a finalidade de automatizar funções de implantação, redimensionamento, atualização, agendamento, movimentação, replicação e demais operações com quaisquer tipos de contêineres (seja *Docker*, LXC, *Rocket*, entre outros) em um *cluster* sem que haja

perda de dados por indisponibilidade de serviço. Já considerando serviços ou aplicações Web em unidades de contêiner distintos, ele tem a capacidade de mover, duplicar, remover contêineres como caminho para lidar com as mais variadas cargas de trabalho, garantindo que as condições do cluster sejam sempre correspondentes à demanda.

Com o *Kubernetes*, é possível executar, de forma eficiente e rápida, dimensionamento de infraestrutura e aplicações em tempo real conforme demanda, implantação fácil de novas aplicações ou microsserviços, otimização promovendo a utilização somente dos recursos de *hardware* necessários onde está hospedado. Pode-se dizer que é uma ferramenta portátil levando em conta que ele pode ser executado tando em nuvem pública quanto privada, uma vez que é independente de implementação de contêiner. Tem a capacidade de suportar determinadas funcionalidades em seu escopo, tais como montagem de sistemas de arquivos, replicação de instâncias de contêineres, balanceamento de carga, monitoramento de recursos, escalabilidade horizontal[1], verificação do funcionamento pleno das unidades de contêiner, entre outros.

Em seu escopo, o *Kubernetes* traz especificações importantes para compreender sua forma de trabalho. Primeiramente deve-se mencionar que o *Kubernetes* tem um ponto principal no *cluster*, chamado *Kubernetes Master*. Considerando um *cluster* com o *Kubernetes Master* e os demais *nodes* em operação, podem-se ainda citar: os *Pods*: são contêineres ou grupos de contêineres de um mesmo tipo de aplicação ou serviço. Esses *Pods*: identificados através de *labels*, organizando então a solicitação e alocação dos recursos. Devem ser levados em consideração também os seletores, que são ordenadores ou selecionadores de etiquetas (*labels*) que verificam a quais *Pods* devem ser direcionadas as demandas do balanceamento de carga. Importante também mencionar a função de controle de replicação (*Replication Controller*), um *serviço interno* que permite quantificar ou escalar cada tipo de contêiner replicando e recriando *Pods* ou unidades de contêiner conforme definido, de acordo com a demanda ou mediante à falha de alguma unidade de *Pod*.

É no controle de replicação que são configurados a quantidade de contêineres ou *Pods* e os tipos de contêineres que devem ser executados, levando em consideração a interessante possibilidade de fazer com que o balanceamento de carga abranja as novas unidades de contêiner ou *Pods* que são iniciadas, rapidamente demandando requisições à mesmas.

A implantação com Kubernetes também permite que um *cluster* possa ser dividido em ambientes, como homologação e produção, por exemplo, através de isolamento de ambientes identificados por *namespaces*, podendo discriminar recursos computacionais

[1] A criação ou implantação de mais unidades de virtualização, sejam essas unidades máquinas virtuais ou contêineres virtuais, já organizando um balanceamento entre as mesmas e considerando que essas unidades trabalhem em prol do mesmo ambiente, trabalhando em *cluster* e provendo os mesmos serviços.

para os ambientes. Portanto, um mesmo *cluster* de máquinas, físicas ou virtuais, podem compartilhar vários projetos ou ambientes de aplicações (multi-hospedagem ou *multi-tenant*). E, para acesso externo aos contêineres, é possível também atrelar determinada faixa de IP para um controlador de replicação graças aos chamados *Services*, que na prática definem endereços IP para as instâncias de Pods.

Outras opções de orquestradores mostraram-se interessantes como o *Apache Mesos* (The Apache Software Foundation, 2017). Distingue-se do Kubernetes por abstrair os recursos computacionais como volumes (*pools*) a serem geridos pelo orquestrador. Falando mais especificamente de contêineres, o *Docker* também traz consigo uma ferramenta que pode gerenciar contêineres e fazer a orquestração. Pode ser uma boa escolha por já fazer parte da *API* do *Docker*, eximindo o administrador de configurar um mecanismo de orquestração a parte. O *Docker Swarm* tem alta disponibilidade utilizando também uma construção com definições de mestre e demais nós e é desenvolvido na mesma linguagem do *Kubernetes*, porém, não conta com um balanceamento de carga externo e também não faz a escalabilidade automática, dependendo de outras ferramentas para autoescalonamento ou realizar tal função de alguma maneira manual.

Optou-se, portanto, pela escolha do *Kubernetes* pelo fato de oferecer a migração dos atuais serviços de máquinas virtuais, dada a sua complexidade (versões, interoperabilidade, integração com LDAP e DNS, entre outros) por um sistema centralizado de orquestração de contêineres e registro de eventos (*logs*). Reduz-se, na prática, a complexidade do ambiente ao remover a camada do sistema operacional físico ou virtualizado.

## 2.4 Rede Virtual para os Contêineres

Para otimizar a infraestrutura, recomenda-se a inserção de um protocolo responsável por escalar uma rede virtual entre os contêineres, viabilizando a comunicação entre os mesmos através de funções como endereçamento de unidades de contêiner. Uma possível solução se dá através do uso de *VXLAN*, virtualizando redes sobre IP (MAHALINGAM et al., 2016).

O *VXLAN* (*Virtual Extensible Local Area Network*) consiste em um protocolo onde é implementada uma camada de enlace virtual sobre uma camada de rede atribuindo, às inúmeras unidades de virtualização, endereçamento através do identificador de rede *VXLAN*. As redes implementadas podem ser denominadas segmentos *VXLAN* e carregam informações de identificação destes segmentos e, consequentemente, por criar redes virtuais, necessitam de um roteamento virtual. Um pacote original enviado por um contêiner, por exemplo, é encapsulado em um cabeçalho *VXLAN* incluindo identificador do segmento *VXLAN* que o contêiner em questão pertence.

O preenchimento do quadro com informações de identificação é feito de forma

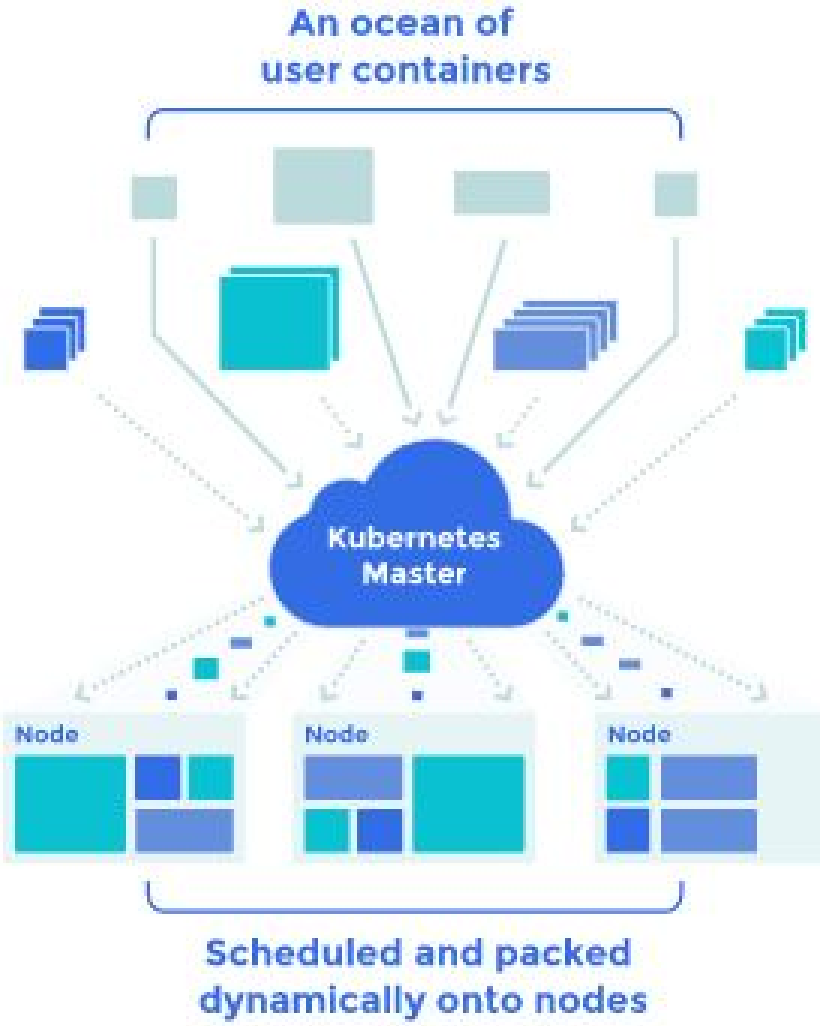


Figura 5 – An ocean of containers. Kubernetes.

dinâmica quando ocorre a comunicação entre duas máquinas virtuais, por exemplo, onde o roteamento virtual atribui aos pacotes um cabeçalho referente ao *VXLAN*, com informações de endereço de origem e destino. Estes fatores resultam em uma fácil escalabilidade de *VLANs* para as unidades de virtualização, saltando de 4094 para mais de 16 milhões de redes virtuais possíveis.

# 3 Desenvolvimento da Proposta

A camada de infraestrutura atual está implantada promovendo os serviços em aplicações Web no modelo conhecido como SaaS (*Aoftware-as-a-Service*). Portanto, são demandados servidores de aplicação virtualizados para os variados serviços e suas ferramentas dependentes para o funcionamento. Pode-se afirmar que a infraestrutura possui diversas unidades de virtualização, as quais são gerenciadas atualmente através do projeto *OpenStack*. O *Openstack* pode ser definido como um sistema operacional projetado para nuvem capaz de administrar uma grande quantidade de recursos computacionais como armazenamento, memória, rede e processamento em determinada infraestrutura. Por ser uma ferramenta altamente flexível e que permite a construção de qualquer tipo de infraestrutura. Trabalhar em *OpenStack* pode exigir grande investimento de esforço e tempo nas atividades de implantação e atualização de serviços ou aplicações e verificação de falhas, atividades estas que devem ser automatizadas seguindo o princípio de contêiner como um serviço com gerenciamento dos contêineres

Em relação ao novo modelo de infraestrutura proposto para as aplicações do ambiente virtual do câmpus, em decorrência das inúmeras vantagens anteriormente mencionadas, tais como fácil portabilidade de aplicações, flexibilidade, velocidade e confiabilidade na implantação e atualização de *webapps*, fácil escalabilidade, infraestrutura redimensionável em conformidade com a carga de trabalho, alto nível de abstração, entre outros; a solução baseada em contêineres permite que não se tenha a preocupação com dimensionamento de infraestrutura, apenas o esforço de automatizar a criação de instâncias de contêiner nos mais variados serviços tomando como base leituras na carga das demandas.

A solução pode ser implantada diretamente na infraestrutura física anteriormente mencionada, ou através de máquinas virtuais, colocando-as em rede e configurando as mesmas para que se caracterize o funcionamento em modo *cluster*. Adotou-se o sistema operacional CoreOS devido para auxiliar no processo de implantação da infraestrutura e, assim, manter o foco nos serviços em contêiner.

Com o sistema já em execução, é necessária a instalação dos pacotes do contêiner *Docker* em cada uma das máquinas, bem como a configuração dos *Dockerfiles* de forma que sejam montados corretamente os contêineres com as imagens (*Docker Images*) dos conteúdos, inicializando assim os mais variados serviços, como sistemas de arquivos distribuídos com o *Ceph*, agendamento de tarefas com a *cron*, resolução de nomes de domínio, *Varnish* cache, balanceamento de carga através do *Haproxy*, base de dados em *MySQL*, servidor web através do *Nginx* e demais aplicações web do campus, portal do aluno, ambiente de *wiki* do campus, sistema do *moodle* e demais serviços contidos em cada

unidade de contêiner podendo ser movidos, replicados, escalados, clonados, removidos, entre outras ações.

Tais ações são executadas com a ferramenta administradora de contêineres, que será o *Google Kubernetes*. Os pacotes do *Kubernetes* também devem ser instalados em todas as máquinas virtuais dispostas no *cluster*, para que o serviço sempre esteja disponível e possa autoescalar-se em ocasiões de perdas de unidades de contêiner ou *Pods* ou até mesmo indisponibilidades nos servidores com *CoreOS*. Através do controlador de replicação podem ser definidos um número máximo e mínimo de contêineres ou *Pods* e como eles devem agir em momentos de falhas ou necessidade de expansão, recebendo parâmetros de endereçamento através dos *services* ou de qualquer outra ferramenta auxiliar.

Com o ambiente disponível, são importantes alguns procedimentos que possam atestar o comportamento da hospedagem na nova infraestrutura perante falhas ou redimensionamento. Existem algumas ferramentas que podem simular milhares de requisições ou acessos a determinada página. Um exemplo de ferramenta que pode auxiliar no processo é o *LoadImpact* que também simula várias requisições a um *site* e permite gerar relatórios, o que facilita a identificação e correção de pontos de falha. Ainda com um determinado número de contêineres em execução e inúmeras requisições em andamento, testa-se também a remoção de contêineres ou implantação de aplicações ou serviços com defeitos propositais, para que seja comprovada a função de redimensionamento e restabelecimento da infraestrutura e o comportamento da mesma, bem como atividades de monitoramento do ambiente em plena execução.

## 3.1 Implantação da Nova Infraestrutura

Reiterando informações anteriormente citadas, a infraestrutura conta com três elementos principais para implementação, que tratam-se da distribuição utilizada nos servidores físicos, o tipo de contêiner que conterá cada serviço e a ferramenta responsável pelo escalonamento e gerenciamento das unidades de encapsulamento baseadas em contêiner.

Iniciando a preparação do *cluster*, foram disponibilizados três servidores físicos, conforme mencionado no item 1.1.

Foi também definido o endereçamento para acesso externo aos servidores, mediante consulta prévia e identificação de endereços IP provenientes da faixa disponibilizada pela Rede Nacional de Ensino e Pesquisa (RNP). Para o *coreos-0* foi designado o endereço para acesso externo 200.135.37.93, para o *coreos-1* o IP 200.135.37.94 e o endereço IP 200.135.37.95 para o *coreos-2*.

Ainda que se tenham todos estes recursos computacionais, é possível montar os cenários de testes utilizando somente uma máquina virtual com as ferramentas instaladas,

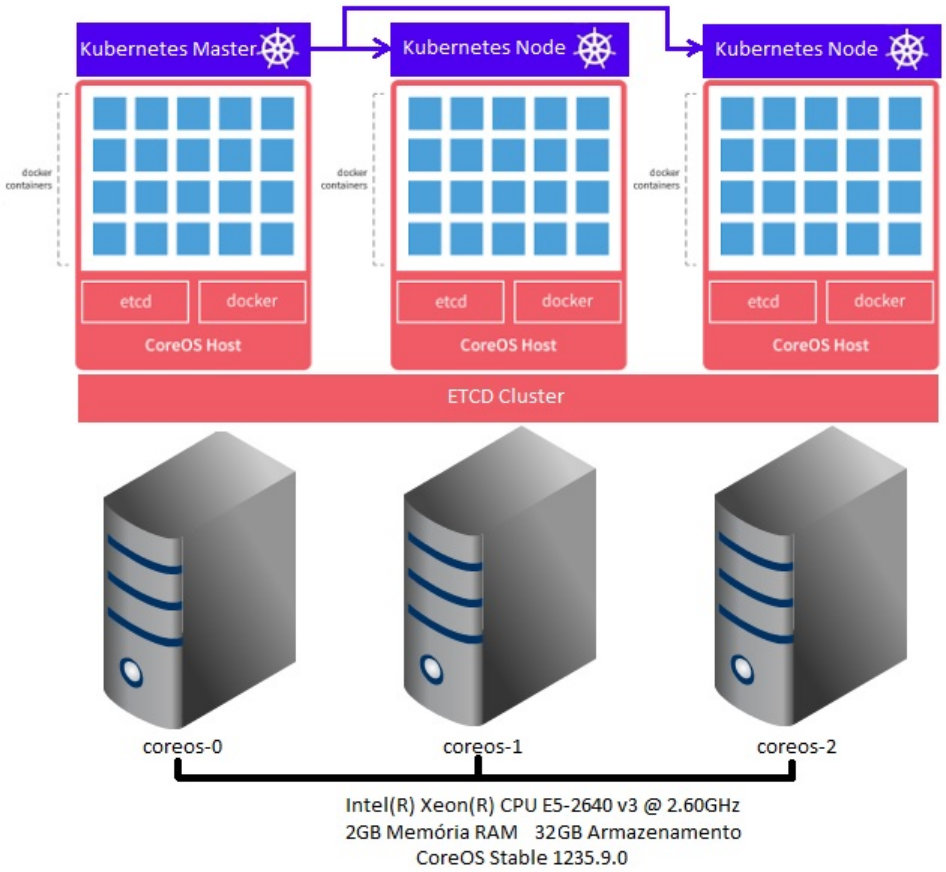


Figura 6 – Topologia da infraestrutura. Fonte: elaborado pelo autor.

caracterizando uma infraestrutura relativamente enxuta perante outros métodos.

A instalação do sistema operacional é feita mediante uso do arquivo *ISO* em cada um dos servidores, utilizando a imagem oriunda da documentação oficial do *CoreOS*, e um *script*, também disponibilizado pela entidade, limpando todos os dados atuais do disco e realizando a instalação do *CoreOS* em sua versão 1235.9.0. Esta distribuição é então instalada e ocupa em torno de 150MB de espaço em armazenamento do servidor (CoreOS, Inc., 2016).

Com o *CoreOS* em execução nos servidores, foi necessária a instalação de algumas ferramentas para dispor os mesmos em modo *cluster*, reconhecendo uns aos outros e compartilhando dados a respeito dos hosts e dos contêineres definindo um cluster construído de acordo com a Figura 6.

## 3.2 Ferramentas para o *Cluster*

Pode-se iniciar a implantação das ferramentas auxiliares promovendo a instalação do serviço que será responsável por armazenar e alinhar dados entre os contêineres criando um *cluster* de unidades dos mesmos, trata-se do `etcd`.

Basicamente é criado um *script* com métricas de endereçamento para dimensionar o *cluster Etcd*, como informações dos elementos do cluster, porta, *hostname*, estado inicial, entre outros. Este *script* deve ser interpretado pelo *cloud-config* para que possa ser executado pelo *systemd* (CoreOS, Inc., 2016).

O arquivo relativo a este *script* é, neste caso, o *10-environment.conf* e traz dados a respeito do endereço específico para este terminal *Etcd* e dos demais elementos deste *cluster* com seus respectivos *hostnames*, que serão os *endpoints* procurados pelo *Kubernetes* ao criar unidades. Também podem ser definidas outras métricas de ambiente em *cluster*, como pode ser visto através do seguinte conteúdo:

```
[Service]
Environment="ETCD_ADVERTISE_CLIENT_URLS=http://200.135.37.93:2379"
Environment="ETCD_INITIAL_ADVERTISE_PEER_URLS=http://200.135.37.93:2380"
Environment="ETCD_INITIAL_CLUSTER=coreos-0=http://200.135.37.93:2380,
coreos-1=http://200.135.37.94:2380,coreos-2=http://200.135.37.95:2380"
Environment="ETCD_INITIAL_CLUSTER_STATE=new"
Environment="ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379"
Environment="ETCD_LISTEN_PEER_URLS=http://200.135.37.93:2380"
Environment="ETCD_NAME=coreos-0"
```

Assim que iniciado, o *Etcd* já recebe um valor que o identifica no *cluster*, que é o seu ID. Caso o *cluster Etcd* contenha mais de um elemento em sua topologia, pode ser seguido o mesmo padrão de instalação e criação de *script* para os demais membros, atentando-se somente a detalhes de endereçamento para que não haja nenhum tipo de conflito ao iniciar o cluster.

Considerando um *cluster* com dois ou três membros, por exemplo, o *Etcd* permite que, assim que os membros são todos inicializados, possam ser identificados uns pelos outros através de comandos do próprio *Etcd*.

Para atestar tais funcionalidades, ao iniciar o serviço nos membros do *cluster*, é possível executar o comando:

```
etcdctl member list
```

e tendo como resposta todos os membros do *cluster* com informações de identificador (ID), *hostname* e endereços, como pode ser visto no retorno do comando a seguir:

```
core@coreos-0 ~ $ etcdctl member list
a35c52a001242d4d: name=coreos-0 peerURLs=http://200.135.37.93:2380
clientURLs=http://200.135.37.93:2379 isLeader=true
```

```
d02fadedace63993: name=coreos-2 peerURLs=http://200.135.37.95:2380
clientURLs=http://200.135.37.95:2379 isLeader=false
f9c0346cd38c1680: name=coreos-1 peerURLs=http://200.135.37.94:2380
clientURLs=http://200.135.37.94:2379 isLeader=false
```

Pode-se, também, adicionar, remover membros ou realizar *backup* através do mesmo comando.

Além de listar os elementos do *cluster*, é possível que o *Etcd* ainda faça uma verificação nos demais membros identificados a fim de confirmar se os mesmos estão ativos, ou seja, checar a disponibiliadde do *cluster* através do comando *etcdctl cluster-health* tendo, então, a resposta com informações como pode ser visto através da saída a seguir:

```
core@coreos-0 ~ $ etcdctl cluster-health
member a35c52a001242d4d is healthy: got healthy result from
http://200.135.37.93:2379
member d02fadedace63993 is healthy: got healthy result from
http://200.135.37.95:2379
member f9c0346cd38c1680 is healthy: got healthy result from
http://200.135.37.94:2379
cluster is healthy
```

Com o serviço do *Etcd* em execução, fez-se necessária a implementação do serviço referente ao contêiner, neste caso o *Docker*.

## 3.3 Instalação do Docker

Assim como já fora dito, a escolha da distribuição Linux CoreOS visa facilitar a implantação da infraestrutura básica do cenário. O gerenciador de contêineres Docker, por exemplo, já vem instalado. Foi utilizada, para os testes, a versão 1.12.6, considerada estável à época.

Com o contêiner *Docker* instalado, pode-se iniciar reconhecendo as funções do comando *docker*, que permite atividades de criar, remover, reiniciar, listar, pausar contêineres, entre outras funções.

Iniciaram-se os testes rodando um contêiner com um serviço de blog, neste caso, a aplicação *Wordpress* encapsulada, realizando a busca da imagem no repositório oficial, executando para tal:

```
docker pull wordpress
```

Fez-se assim, o descarregamento da imagem e obter como resposta as várias camadas da imagem, como pode ver a seguir um instante do progresso da operação:

```
core@coreos-0 ~ $ docker pull wordpress
Using default tag: latest
latest: Pulling from library/wordpress

5040bd298390: Already exists
568dce68541a: Downloading
[==================================>   ] 73.84 MB/77.61 MB
6a832068e64c: Download complete
3b0f3d176a5b: Download complete
20cc248a5690: Download complete
6ff565538ee6: Download complete
9f1077228581: Download complete
57359f144a19: Download complete
9754ef36b033: Download complete
e156df35b624: Download complete
df09daa2224a: Download complete
bfa0d8031302: Download complete
58ec6fadb2c7: Download complete
254919ae58d3: Download complete
3fbda78d9add: Waiting
1f4006165e89: Waiting
1989a57f5f0b: Waiting
9d38d20a9daa: Waiting
```

Para este exemplo, um simples docker pull foi executado já buscando no *Docker Hub* as imagens necessárias para a implementação do *Wordpress*, e fazendo o descarregamento das mesmas.

Para este serviço, fez-se necessária também a instalação de uma imagem de contêiner de banco de dados, representada pelo *Mysql*, onde foi obtido via:

```
docker pull mysql
```

E então, foi feito o descarregamento da imagem do serviço de banco de dados *MySQL*, a partir do Dockerhub, como pode ser visto:

```
core@coreos-0 ~ $ docker pull mysql
Using default tag: latest
```

```
latest: Pulling from library/mysql

5040bd298390: Already exists
55370df68315: Pull complete
fad5195d69cc: Pull complete
a1034a5fbbfc: Pull complete
17f3570b42ae: Pull complete
6bf4b16e5339: Pull complete
9700c9731729: Pull complete
f2fea9c5b632: Downloading
[=============>                        ] 42.71 MB/77 MB
2f8101f5336d: Download complete
0dc8f8a1031a: Download complete
a1b9627588c7: Download complete
```

Utilizando outro comando de início para contêineres *Docker*, foi possível subir o contêiner de *mysql* já definindo o nome do contêiner, a versão mais recente da imagem e até mesmo a senha de root para o banco de dados, recebendo como retorno o *CONTAINER ID* completo para este contêiner:

```
core@coreos-0 ~ $ docker run --name docker-mysql -e
MYSQL_ROOT_PASSWORD=docker -d mysql:latest
c1ae4424374fed1515eeea332e8acb2599d60b7b45dfe74c9047f816a6ccd504
```

O banco de dados já está em execução e com um identificador de contêiner atribuído, bem como informações de tempo de criação, status, entre outras métricas que podem ser confirmadas ao rodar o comando de listar os contêineres em execução:

```
core@coreos-0 ~ $ docker ps
CONTAINER ID    IMAGE           COMMAND                  CREATED
c1ae4424374f    mysql:latest    "docker-entrypoint.sh"  27 seconds ago
STATUS          PORTS       NAMES
Up 26 seconds   3306/tcp    docker-mysql
```

Com o banco de dados em operação, foi então iniciado um contêiner com a imagem do *Wordpress* previamente adquirida, atribuindo um nome à mesma, definindo as portas para o servidor e para o contêiner e criando um link entre contêineres, neste caso, atrelando a imagem de *mysql* anteriormente criada como banco de dados para o serviço do wordpress e já recebendo também um identificador do contêiner para o serviço de blog:

```
core@coreos-0 ~ $ docker run --name docker-wordpress --link
docker-mysql:mysql -p 9080:80 -d wordpress
66adf917d8139e7552227ad542e3f151090302daf25fd72822bbad40dbe2672d
```

O contêiner de *Wordpress* já está iniciado, com portas atribuídas e é listado através de um *docker ps* juntamente com o contêiner do *mysql*:

```
core@coreos-0 ~ $ docker ps
CONTAINER ID    IMAGE        COMMAND                  CREATED
66adf917d813    wordpress    "docker-entrypoint.sh"  2 minutes ago
c1ae4424374f    mysql:latest "docker-entrypoint.sh"  3 minutes ago
STATUS          PORTS                    NAMES
Up 2 minutes    0.0.0.0:9080->80/tcp    docker-wordpress
Up 3 minutes    3306/tcp                 docker-mysql
```

Após isso, em uma outra máquina na rede, já é possível configurar o *Wordpress* encapsulado em contêiner abrindo o navegador e acessando *http://200.135.37.93:9080* - a porta 9080/TCP é atribuída por padrão na ausência de um parâmetro que o altere.

Ainda seguindo os testes com aplicação manual de contêineres, foi executado uma nova unidade de contêiner, também aproveitando o contêiner de mysql como banco de dados, para a implantação de um serviço de cliente de correio eletrônico, neste caso, o *Roundcube*.

Este exemplo foi adotado pois é possível configurar uma série de parâmetros e mapeamentos para um contêiner e para acesso ao mesmo através de um único comando. Neste caso, primeiramente foi feito o download da imagem docker para o serviço de webmail através do *Docker Hub*, rodando o seguinte comando:

```
docker pull classcat/webmail
```

Então é feito o download da imagem do webmail:

```
core@coreos-0 ~ $ docker pull classcat/webmail
Using default tag: latest
latest: Pulling from classcat/webmail

e190868d63f8: Download complete
909cd34c6fd7: Download complete
0b9bfabab7c1: Download complete
a3ed95caeb02: Download complete
```

```
db723e1865b3: Download complete
06c47a7e289d: Download complete
5fdc105f731f: Download complete
d8340a2fc8fa: Download complete
0c5e6fe95a69: Download complete
3e80f04fe461: Download complete
9aa191891596: Download complete
6a9921dee9a3: Download complete
2965f722c5ac: Download complete
bbd43fca3359: Download complete
354516acb488: Download complete
Digest: sha256:203339c10f4067d5ee3fcf9b08a1a973838a02e3790bf
6034c8760b56c3f6698
Status: Image is up to date for classcat/webmail:latest
```

Uma instância da imagem recém adquirida do contêiner é posta em execução. Neste caso, foram definidos nome para o contêiner, porta SSH, porta de acesso web, atribuição ao contêiner de banco de dados, senha de administrador (usuário `root` para a aplicação *Web*), chave pública de autenticação, configurações de servidor, usuário e senha para o servidor SMTP e idioma. Para tais definições, seguiu-se com o seguinte comando:

```
core@coreos-0 ~ $ docker run --name docker-roundcube -p 2022:22 -p
10080:80 --link docker-mysql:mysql -e ROOT_PASSWORD=wordpress -e
SSH_PUBLIC_KEY="ssh-rsa docker" -e
DEFAULT_HOST=dockermail.roundcube.com -e
SMTP_SERVER=dockermail.roundcube.com -e SMTP_USER=docker -e
SMTP_PASS=dockerpass -e LANGUAGE=en_US -d classcat/webmail
1f6718675fe64b3bb699045b2578e14840a44256c5a767576c49b99c20c88e4d
```

Na sequência, já é possível ver o contêiner do webmail listado com os demais contêineres:

```
core@coreos-0 ~ $ docker ps
CONTAINER ID  IMAGE            COMMAND                  CREATED
1f6718675fe6  classcat/webmail "/bin/sh -c '/opt/cc-" 3 minutes ago
66adf917d813  wordpress        "docker-entrypoint.sh" 20 minutes ago
c1ae4424374f  mysql:latest     "docker-entrypoint.sh" 22 minutes ago
STATUS          PORTS
Up 3 minutes    0.0.0.0:2022->22/tcp, 0.0.0.0:10080->80/tcp
Up 20 minutes   0.0.0.0:9080->80/tcp
```

```
Up 22 minutes   3306/tcp
NAMES
docker-roundcube
docker-wordpress
docker-mysql
```

Para verificação de todas as imagens previamente baixadas e instaladas, bem como algumas informações de ID, tempo de criação e tamanho total das mesmas, pode-se executar o seguinte comando:

```
core@coreos-0 ~ $ docker images
REPOSITORY       TAG       IMAGE ID      CREATED       SIZE
wordpress        latest    109633df95f5  8 days ago    400.1 MB
mysql            latest    7666f75adb6b  8 days ago    405.6 MB
classcat/webmail latest    3cf8e9a484a1  20 months ago 480.2 MB
```

Para depuração de falhas e acompanhamento de determinado serviço, é possível também analisar os arquivos de registros de eventos (*logs*) de cada serviço isolado em cada contêiner. Um exemplo pode ser visto ao ler os *logs* do contêiner `mysql`, neste caso, foi executado o seguinte comando:

```
core@coreos-0 / $ docker logs -f docker-mysql
2017-02-05T20:12:44.438128Z 0 [Warning] 'proxies_priv' entry '@
root@localhost' ignored in --skip-name-resolve mode.
2017-02-05T20:12:44.439750Z 0 [Warning] 'tables_priv' entry
'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
2017-02-05T20:12:44.443815Z 0 [Note] Event Scheduler: Loaded 0
events
2017-02-05T20:12:44.443895Z 0 [Note] Executing 'SELECT * FROM
INFORMATION_SCHEMA.TABLES;' to get a list of tables using the
deprecated partition engine. You may use the startup option
'--disable-partition-engine-check' to skip this check.
```

Ainda em implantação não automatizada, percebeu-se certa rapidez e praticidade nos procedimentos de instalação de serviços Web, bem como configuração e definições de parâmetros dos mesmos, visto que nas implantações com serviços em máquinas virtuais, eram exigidos mais procedimentos como descarregamentos de pacotes - esses específicos por distribuição e/ou versão, uma quantidade maior de comandos executados e, consequentemente, mais tempo demandado para as tarefas.

Em poucos instantes foi possível iniciar 3 serviços encapsulados em forma de contêineres Docker. Estes serviços normalmente costumam consumir bastantes recursos recursos computacionais. Foram eles um banco de dados *mysql*, um site *Wordpress* e um servidor de webmail *Roundcube*, já configurados e em prontos para execução e utilização. Normalmente, um serviço como o *MySQL* demanda certo tempo no procedimento de instalação, visto que é necessário fazer o download do pacote e instalação, que carrega atividades de criação de usuário, senha, criação de tabelas, entre outros. Um serviço de *Wordpress* pode levar ainda mais tempo exigindo, além de descarregamento e instalação, alterações em arquivos de configurações, atribuição manual a um banco de dados, entre outras atividades.

Dependendo do serviço ou da aplicação que será encapsulada, o método pode não ser tão vantajoso, pois pode ser necessário também o isolamento de arquivos de bibliotecas junto com a aplicação, fator que pode tornar uma unidade de contêiner com mais conteudo e, consequentemente, mais pesada: processamento extra para tráfego inter-contêiner, considerando o isolamento das aplicações e abstração de rede virtual sobre UDP (VXLAN).

## 3.4 Testes com Kubernetes

Inicialmente, já considerando o *cluster* `etcd` em funcionamento e compatilhando dados de contêineres entre os sevidores, é feita a implantação de VXLAN com a aplicação *Flannel*, que irá prover o tráfego criando uma rede entre os contêineres ou grupos de contêineres e definindo, através dos*scripts* da ferramenta, os servidores do *cluster*, aqui chamados de *endpoints* e o local considerado nó principal para a interface do *Flannel*. Foi também necessário garantir, através outras métricas, que o *Flannel* funcionasse com os parâmetros acima mencionados.

Para que seja possível rodar a API do *Kubernetes*,bem como executar suas funcionalidades, é necessária a criação de um serviço gerenciador dos contêineres mínimos de controle:

1. *API server*;
2. *Controller Manager*;
3. *Scheduler*;
4. *Proxy*.

O serviço *Kubelet*, portanto, é respońsável por garantir o funcionamento desses serviços mínimos, a fim de demandar tarefas de criação de *Pods*, que são os grupos de contêineres

para determinados serviços. O principal ponto dessa instalação, que também trata-se da ferramenta mais utilizada, é o *kubectl*. Ele é uma CLI (*Command Line Interface*) para interações com a servidor API do Kubernetes. Em termos, o *kubectl* também trata-se de um comando para o *Kubernetes* que tem atividades de checagem de *cluster*, implantação de *pods* e outras funcionalidades similares ao *docker*, porém em *cluster*.

Inicia-se a implantação do *kubectl* fazendo o download a partir de um repositório, na documentação oficial da ferramenta, através de um cliente HTTP como o cURL:

```
core@coreos-0 ~ $ curl -LO
https://storage.googleapis.com/kubernetes-release/
release/$(curl -s https://storage.googleapis.com/
kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
% Total    %   Received  %    Xferd  Average Speed   Time     Time
                                      Dload   Upload  Total    Spent
70 48.0M   70   34.0M    0     0     3704k    0     0:00:13  0:00:09
Time     Current
Left     Speed
0:00:04  5805k
```

Após procedimento de descarregamento e configuração de algumas variáveis mais específicas com o auxílio da documentação oficial, já é possível utilizar os comandos do *kubectl* para checar valores e dados do cluster. É possível, por exemplo, ver os demais nós do cluster:

```
core@coreos-0 ~ $ kubectl get nodes
NAME       STATUS    AGE
coreos-1   Ready     213d
coreos-2   Ready     213d
```

Neste ponto dos testes, a ferramenta mostrou-se mais intuitiva ao realizar o procedimento de inicialização mútipla de *pods* de um servidor Web NGINX, por exemplo. Inicialmente, foi executado o comando

```
kubectl run
```

similar ao comando de inicialização de contêineres do *docker*, definindo um nome para a *pod*, buscando a imagem do *nginx*, atribuindo uma porta de acesso ao serviço e a quantidade de réplicas, através do comando:

```
core@coreos-0 ~ $ kubectl run my-nginx \
```

```
                    --image=nginx --replicas=2 --port=80
deployment "my-nginx" created
```

Após iniciar os contêineres de nginx, é possível ver as réplicas já criadas, com um comando:

```
core@coreos-0 ~ $ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
kube-proxy-coreos-1           1/1       Running   23         213d
kube-proxy-coreos-2           1/1       Running   24         64d
my-nginx-2494149703-i9wr7     1/1       Running   0          20s
my-nginx-2494149703-mg6g4     1/1       Running   0          20s
```

Informações de quantidade de contêineres desejada, quantidade atual de contêineres da *pod*, contêineres disponíveis e tempo de contêiner em execução (*uptime*) para o NGINX recém iniciada, podem ser confirmadas executando:

```
core@coreos-0 ~ $ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   2         2         2            2           8m
```

Após os procedimentos iniciais, fez-se necessária a publicação do serviço para acesso. Neste caso, deve ser exposta a implantação (*deployment*) iniciada, definindo a porta padrão 80 para acesso, e o tipo de comportamento como balanceador de carga (*load balancer*), através de:

```
core@coreos-0 ~ $ kubectl expose deployment my-nginx \
                    --port=80 --type=LoadBalancer
service "my-nginx" exposed
```

O *kubectl* nos permite também checar os serviços que estão em execução no *Kubernetes*, bem como IP (diferente da máquina que hospeda o contêiner) e porta exposta (para acesso externo ao *cluster*):

```
core@coreos-0 ~ $ kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.1.0.1       <none>        443/TCP   227d
my-nginx     10.1.203.200   <pending>     80/TCP    7s
```

Se for o caso de remover a implantação criada, o comando

```
kubectl delete deployment my-nginx
```

remove o *deployment* para a aplicação especificada por nome.

Ainda utilizando o servidor Web NGINX como exemplo, novamente percebe-se maior facilidade na implantação de um *cluster*. Considerando três máquinas virtuais com seus sistemas operacionais próprios e tendo a necessidade de instalar o servidor web em ambas, o NGINX também exige, em todos os terminais, procedimentos de descarregamento da imagem e instalação, alterações em arquivos de configuração para definição de métricas como definição de diretórios onde as URLs ficam alocadas, definição de conteúdo estático, como `index.html`, e definição de diretório base para o serviço WEB em si.

Com a utilização do Kubernetes executando contêineres e *pods*, é possível fazer a implantação de várias unidades de virtualização executando *nginx* através de um mesmo comando, no mesmo sistema operacional, já definindo portas de acesso e habilitando o balanceamento de carga para as unidades criadas, funcionalidades que facilitam e economizam tempo em uma implantação de *web services*.

Procederam-se os testes aplicando a funcionalidade de autoescalabilidade à um contêiner de *Apache*, neste caso a imagem *php-apache*. Foi testada a implantação de uma escalabilidade horizontal, no qual um serviço é replicado algumas vezes, dentro do mesmo cluster, a fim de atender à carga de utilização.

Inicialmente, foi implantada uma imagem de exemplo do *php-apache* proveniente da documentação oficial, atribuindo à mesma uma liberação de porta para publicação e o quanto de processamento será utilizado por parte das réplicas, através do seguinte comando

```
core@coreos-0 ~ $ kubectl run php-apache \
                  --image=gcr.io/google_containers/hpa-example \
                  --requests=cpu=200m --expose --port=80
service "php-apache" created
deployment "php-apache" created
```

Após isso, foi definida a autoescalabilidade através do argumento *autoscale*, de maneira simples e intuitiva, definindo para qual imagem ela irá autoescalar-se e até mesmo parametrizando o percentual de processamento que as réplicas irão utilizar e os valores mínimo e máximo de réplicas para este conjunto de contêineres, executando para tal:

```
core@coreos-0 ~ $ kubectl autoscale deployment php-apache \
                  --cpu-percent=60 --min=2 --max=20
```

```
deployment "php-apache" autoscaled
```

Após isso, é possível ver os parâmetros do autoescalonamento criados, com um mínimo de 2 réplicas e um máximo de 20, utilizando 60% de processamento:

```
core@coreos-0 ~ $ kubectl get hpa
NAME       REFERENCE             TARGET CURRENT   MINPODS MAXPODS AGE
php-apache Deployment/php-apache 60\%   <waiting> 2       20      13s
```

É chegado então ao momento de acrescentar uma carga de acessos, a fim de atestar o redimensionamento da infraestrutura, neste caso, optou-se por utilizar gerador de carga, chamado *load-generator*, que demanda uma grande quantidade de acessos à aplicação exigindo bastante da mesma, que, em poucos instantes, autoescala em quantidade de contêineres para atender tal carga através do comando:

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
```

E, então, validando os serviços em implantação: ver a seguir:

```
core@coreos-0 ~ $ kubectl get deployments
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
php-apache   11        11        11           9           1m
```

Após a carga ter sido aplicada e finalizada, percebeu-se que a infraestrutura voltou a autoescalar, reduzindo seu tamanho através da remoção contêineres que não seriam mais utilizados, como pode ser visto a seguir:

```
core@coreos-0 ~ $ kubectl get hpa
NAME       REFERENCE             TARGET CURRENT   MINPODS MAXPODS AGE
php-apache Deployment/php-apache 60\%   <waiting> 2       20      2m
```

Com isso, foi implantado, de forma fácil e prática, um autoescalador capaz de redimensionar a infraestrutura virtual de acordo com a demanda, voltando ao seu temanho inicial após o que consideramos o pico de atividades.

# 4 Conclusões

Os experimentos realizados tiveram o objetivo de explanar e trazer certa proximidade com o método de virtualização baseado em contêineres visto que, desde o início da metodologia de testes, mostrou ser uma construção mais intuitiva e que promove uma infraestrutura mais simples, leve e igualmente capaz de hospedar os mesmos serviços que uma infraestrutura baseada em máquinas virtuais hospeda.

Na implantação que considera-se manual, utilizando somente o *docker*, com dois comandos já foi possível iniciar um serviço de webmail, já apontando à uma base de dados e pronto para acesso via web

Automatizando a criação de contêineres com o *kubectl*, foi percebida maior praticidade na implantação de contêineres, desta vez, de forma múltipla. Assim como os comandos de *docker*, o *kubectl* também trouxe grande riqueza de detalhes a respeito dos grupos de contêineres e seus membros.

Considerado o ponto principal dessa infraestrutura, o autoescalador de contêineres mostrou-se uma solução extremamente rápida e prática, tendo fácil configuração e parametrização alocando de forma inteligente novas unidades de contêiner para receber carga maior de trabalho, conforme testes realizados, e voltando ao estado inicial após término da alta demanda como pôde ser percebido.

Orquestrando de forma automática, o administrador fica isento de atividades como subir máquinas virtuais para hospedagem dos ambientes, inicialização ou parada de serviços, adição de unidades de virtualização para atender maior demanda, entre outros

Mesmo sendo uma infraestrutura relativamente leve e mais simples, considera-se uma desvantagem a necessidade de migrar as aplicações para que seja adequada ao método, criando um *Dockerfile* para determinada aplicação ou serviço e gerando os testes, fato este que pode demandar atividades de reescrita em determinadas aplicações para adequar ao *Docker*. Aplicações que dependem de arquivos de configurações, binários ou bibliotecas podem fazer com que um contêiner seja menos leve, atrasando processos de construção e inicialização.

Possíveis temas para projetos futuros envolvendo contêineres podem ser implementações com contêineres encapsulando serviços de VoIP em seu conteúdo, disponibilizando serviços como o *Asterisk* ou *OpenSIPS* de maneira isolada e sob demanda.

Sugere-se também uma implementação de contêineres em utilizando outras ferramentas, como o *LXC*, o *Rocket*, combinando com os mais variados orquestradores, como o Apache Mesos ou o *Docker Swarm*.

# Referências


CoreOS, Inc. *CoreOS Documentation*. 2015. Acesso em: 12 feb 2016. Disponível em: <https://coreos.com/docs>. Citado na página 21.

CoreOS, Inc. *Getting started with Docker*. 2016. Acesso em: 12 jan 2017. Disponível em: <https://coreos.com/os/docs/latest/getting-started-with-docker.html>. Citado 3 vezes nas páginas 21, 29 e 30.

Docker. *What is Docker?* 2015. Acesso em: 15 feb 2016. Disponível em: <https://www.docker.com/what-docker>. Citado na página 23.

MAHALINGAM, M. et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. 2016. Acesso em: 28 nov 2016. Disponível em: <https://tools.ietf.org/html/rfc7348>. Citado na página 25.

MORAES, G. *Caixa de Ferramentas DevOps: Um guia para construção, administração e arquitetura de sistemas modernos*. [S.l.]: Casa do Código, 2015. ISBN 9788555190827. Citado na página 15.

Netcraft Ltd. *May 2010 Web Server Survey*. 2010. Acesso em: 15 feb 2016. Disponível em: <http://news.netcraft.com/archives/2010/05/14/may_2010_web_server_survey.html>. Citado na página 15.

The Apache Software Foundation. *Mesos Documentation*. 2017. Acesso em: 19 mar 2017. Disponível em: <http://mesos.apache.org/>. Citado na página 25.

The Linux Foundation. *Kubernetes: Production-Grade Container Orchestration*. 2016. Acesso em: 12 feb 2016. Disponível em: <http://kubernetes.io/>. Citado na página 23.

VERMA, A. et al. Large-scale cluster management at Google with Borg. In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France: [s.n.], 2015. Citado 2 vezes nas páginas 19 e 23.