

**UNIVERSIDADE VEIGA DE ALMEIDA – UVA**

**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**DOCKER: UM CANIVETE SUÍÇO, COM APLICAÇÃO EM  
RASPBERRY PI**

**Thiago Soares da Cruz**

**RIO DE JANEIRO**

**2017**

**UNIVERSIDADE VEIGA DE ALMEIDA - UVA**

**THIAGO SOARES DA CRUZ**

Monografia apresentada ao curso de Ciência da Computação da Universidade Veiga de Almeida, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Carlos Alberto Alves Lemos, DSc.

**DOCKER: UM CANIVETE SUÍÇO, COM APLICAÇÃO EM  
RASPBERRY PI**

**RIO DE JANEIRO**

**2017**

**UNIVERSIDADE VEIGA DE ALMEIDA - UVA**  
**BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO**

**THIAGO SOARES DA CRUZ**

**DOCKER: UM CANIVETE SUÍÇO, COM APLICAÇÃO EM  
RASPBERRY PI**

Monografia apresentada ao curso de Ciência da Computação da Universidade Veiga de Almeida, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

APROVADA EM:

CONCEITO: \_\_\_\_\_

BANCA EXAMINADORA:

---

**PROF. Carlos Alberto Alves Lemos, DSc.**

ORIENTADOR

---

**PROF. Miguel Figueiredo Ângelo Zaccur, DSc.**

---

**PROF. Jobson Luiz Massolar. DSc.**

**Coordenação de Ciência da Computação**

Rio de Janeiro, 10 de junho de 2017

*Dedico este trabalho à minha família, em especial a minha mãe, que sempre acreditou no meu potencial, me impulsionando sempre a ir em frente, fazendo desta caminhada, um grande aprendizado e tornando-a possível.*

*E todos aqueles que sempre estiveram comigo diante dos problemas e das adversidades me incentivando sempre a levantar e a lutar, para concluir mais essa etapa da minha caminhada profissional e pessoal.*

*Dedico a Deus, por me permitir continuar nesta reencarnaçāo, por sempre cuidar de mim, para que eu possa trilhar o meu caminho e sempre continuar no caminho da Luz.*

“Conhecer, é poder”

Fonte: Francis Bacon

## AGRADECIMENTO

Faço um agradecimento especial deste trabalho primeiramente a minha mãe, que sempre teve amor, dedicação e garra para poder me ensinar a viver, a passar por todos os obstáculos, sempre me impulsionando a ir em frente e a nunca desistir; ela que esteve comigo em todos os meus momentos de doenças e adversidades, tornando esta minha caminhada possível e cada vez mais me impulsionando e permitindo que eu tomasse as minhas decisões para continuar em frente. Obrigado mãe, é um prazer ser seu filho e ter sido criado com tanto amor e dedicação por você.

Em segundo agradecimento faço a minha amada avó Vilma, ela que sempre apoiou a família em todos os seus piores momentos, que não foram poucos e sempre esteve ao nosso lado, que nunca nos deixou desanimar, que sempre foi o pilar e o exemplo para a vida.

Faço um agradecimento à minha irmã Thiane, que sempre foi um doce de pessoa, sempre esteve junto da família, colaborando e sendo quase como uma segunda mãe para mim; a melhor irmã que poderia ser; te amo muito.

Faço um agradecimento à Deus, que com sua benevolência permitiu a minha reencarnação, me dando como um presente a pessoa maravilhosa que é a minha mãe. Sempre colocou pessoas boas e seres de luz em meu caminho para meu crescimento, aprendizado e proteção, sempre esteve comigo em meus piores momentos me guiando para a luz.

Faço aqui um agradecimento aos meus professores, que sempre me ensinaram muito, alguns foram bem severos e me ensinaram lições muito importantes para a vida pessoal, profissional e acadêmica; e aqueles que também não o foram, não deixaram de me ensinar algo.

Faço um agradecimento a alguns colegas de trabalho: Júlio Bueno, que me ajudou em dúvidas e informações e me autorizou a utilizar parte do seu código para o estudo de caso desta obra; ao Fábio Marinho e ao Leonardo Barros que gentilmente me cederam parte do seu tempo e conhecimento com a ajuda à infraestrutura montada nesta obra; ao Marcelo Iepsen, que com suas dicas de como poderia iniciar e concluir essa obra e a equipe Tsuru de uma empresa de internet de um grupo de mídia, que me permitiu entrevista e acompanhamento de um dos seus dias de trabalho.

Agradeço a todos que contribuíram para a conclusão desta obra, seja com dicas, sugestões, ajuda direta e aos entrevistados que me cederam gentilmente parte do seu tempo para que eu pudesse aumentar o meu conhecimento sobre o assunto.

## **RESUMO**

Nesta obra, abordo o tema da plataforma Docker, com uma aplicação utilizando à arquitetura de serviços em containers. Este tipo de programação separa a aplicação em serviços, da qual a aplicação vai se encaixando como se fosse um bloco de caixas, sendo cada caixa um serviço específico. Estes serviços podem ser providos por módulos ligados a própria aplicação, ou módulos extensores de outras aplicações, que fazem interfaces por meio de *APIs*. Este conceito de serviços para servir aplicações está diretamente ligado ao paradigma programação e processamento em nuvem, da qual cada serviço possa estar fora do da aplicação e ser chamado conforme a sua demanda, programação em micro serviços. Nesta obra utilizo uma infraestrutura própria em Raspberry Pi, como prova de conceito para o tema que estou propondo, usei esta arquitetura para poder comprovar a portabilidade do docker e melhorar o meu estudo de caso, com a possibilidade de execução em infraestrutura própria e distribuída.

**Palavras-Chave:** Docker, IOT, Cloud, Container, Raspberry PI, Cluster, Escalonamento, Aplicação, Serviço, Ruby, Microserviço

## **ABSTRACT**

In this work, I approach the theme of the Docker platform, with an application using the container services architecture. This type of programming separates the application into services, from which the application will fit as if it were a block of boxes, each box being a specific service. These services can be provided by modules connected to the application itself, or extension modules of other applications, that interface through APIs. This concept of services to serve applications is directly linked to the paradigm programming and processing in the cloud, from which each service can be out of the application and be called according to its demand, programming in micro services.

In this work, I use my own infrastructure in Raspberry Pi as a proof of concept for the theme I'm proposing, I used this architecture to test the portability of docker and improve my case study, with the possibility of running on own and distributed infrastructure.

**Keywords:** Docker, IOT, Cloud, Container, Raspberry PI, Cluster, Scheduling, Applications, Service, Ruby, Microservice

## LISTA DE ILUSTRAÇÕES

FIGURA 1: CLOUD .....	20
FIGURA 2: JOSEPH CARL .....	21
FIGURA 3: JOHN MCCARTHY .....	22
FIGURA 4: RAMNATH CHELLAPPA .....	22
FIGURA 5: MODELOS DE IMPLEMENTAÇÃO .....	24
FIGURA 6: NUVEM PÚBLICA .....	25
FIGURA 7: NUVEM PRIVADA .....	26
FIGURA 8: NUVEM COMUNITÁRIA .....	27
FIGURA 9: MODELOS DE IMPLEMENTAÇÃO .....	28
FIGURA 10 - MODELOS DE SERVIÇO .....	31
FIGURA 11 HYPERVISOR HOSPEDADO .....	32
FIGURA 12 - EVOLUÇÃO DO HYPERVISOR .....	33
FIGURA 13 - VIRTUALIZAÇÃO POR CONTAINER .....	33
FIGURA 15 - LXC VS KVM .....	35
FIGURA 16 - DOCKER HUB DO PROJETO .....	47
FIGURA 17 – EXPLORER DO DOCKER HUB .....	47
FIGURA 18 – REPOSITÓRIO OFICIAL DA IMAGEM DO UBUNTU .....	48
FIGURA 19 - DASHBOARD DO PORTAINER .....	52
FIGURA 20 - DASHBOARD DE IMAGENS DO PORTAINER .....	52
FIGURA 21 - DASHBOARD DE VOLUMES DO PORTAINER .....	53
FIGURA 22 - DASHBOARD DE ENGINE DO PORTAINER .....	53
FIGURA 23 - VISUALIZER-ARM FONTE: PRÓPRIO AUTOR .....	56
FIGURA 24 - PLAY WITH DOCKER .....	57
FIGURA 25 - SHELL DO PLAY WITH DOCKER .....	58
FIGURA 26 - TEMPLATE DO PLAY WITH DOCKER .....	59
FIGURA 27 - COMUNIDADE DO DOCKER NO TELEGRAM .....	60
FIGURA 28 - CANAL ANNOUNCEMENTS DO SLACK .....	60
FIGURA 29 - CANAL RANDOM DO SLACK .....	60
FIGURA 30 - DIAGRAMA DE ATIVIDADES DO ESTUDO DE CASO .....	68
FIGURA 31 - DASHBOARD PRINCIPAL DO ESTUDO DE CASO .....	71
FIGURA 32 - DASHBOARD DE PRODUTOS DO ESTUDO DE CASO .....	71
FIGURA 33 - DASHBOARD DE TAGS DO ESTUDO DE CASO .....	72
FIGURA 34 - DASHBOARD DE WHITE LIST DO ESTUDO DE CASO .....	72
FIGURA 35 - DASHBOARD DE USUÁRIOS DO ESTUDO DE CASO .....	72
FIGURA 36 - DASHBOARD DE RESULTADO DE BUSCA DO ESTUDO DE CASO .....	73
FIGURA 37 - DASHBOARD DE FILAS DO SIDEKIQ DO ESTUDO DE CASO .....	73
FIGURA 38 - DASHBOARD DE HISTÓRICO DE BUSCA DO SIDEKIQ DO ESTUDO DE CASO .....	74

## **LISTA DE TABELAS**

TABELA 1- TABELA DE IMAGENS UTILIZADAS NO ESTUDO DE CASO.....	45
TABELA 2 - TABELA DE CONTAINERS UTILIZADAS NO ESTUDO DE CASO .....	49
TABELA 3 - TABELA DE PARÂMETROS UTILIZADO NA MANIPULAÇÃO DOS CONTAINERS.....	50
TABELA 4 - TABELA DE PARÂMETROS UTILIZADOS NA EXECUÇÃO DOS CONTAINERS .....	51

## **LISTA DE ABREVIATURAS E SIGLAS**

PAAS	Plataforma as a Service – Plataforma como serviço
DEPLOY	Implementação de software em ambiente (QA, PROD ou Staging).
QA	Quality Assurance (Ambiente de mensuração de Qualidade/Testes de novas implementações).
PROD	Produção (Ambiente de produção da aplicação, da qual fica acessível externamente, provendo valor para usuário ou serviços – API).
DEV	Ambiente local de desenvolvimento do programador/analista.
DOWN-TIME	Tempo de queda (instabilidade) de uma aplicação ao ser colocada em um ambiente via Deploy e ou erros/falhas apresentadas pela aplicação em produção.
ON-DEMAND	Provisionamento de serviços/recursos sobre demanda de requisição é utilização.
DOCKERHUB	Repositório de imagens do Docker, com imagens dos containers registradas de forma pública pelo criador/administrador da conta.
OPEN-SOURCE	Forma de licenciamento de softwares que não há o pagamento de licenças. Podendo ter regulamentação por alguma organização/comunidade e sendo sua distribuição/utilização livre de encargos.
LIBS	Bibliotecas do Sistema Operacional.
OS	Operation System – Sistema Operacional.
KERNEL	Núcleo do Sistema Operacional.
HOST	Computador ou device conectado a uma rede que pode executar instruções computacionais.
DAEMON	Execução de aplicações em background. Informa o PID da aplicação, mas não bloqueia uma sessão do terminal.
MOUNT POINT	Storage no disco, local que pode ser utilizado para guardar dados de execução do container.
CHOWN	Comando Linux que define permissões escrita/leitura para pastas e arquivos dentro do diretório.

STACK TRACE	Trecho de saída da execução de um comando e/ou erro de execução do software.
CMD	Abreviação de Command, execução de um comando dentro do container.
UI	User Interface - Interface de usuário, interfaces que facilitam a interação do usuário com o sistema.
CI	Continuos Integration – Integração Contínua, são softwares para automatizar o deploy das aplicações nos ambientes.
SCHEDULE	Agendamento e execução de tarefas em background. Software que executa instruções de tarefas agendadas em Daemon, sem ter um terminal e/ou seção vigente.

# SUMÁRIO

<b>INTRODUÇÃO .....</b>	<b>16</b>
<b>2 METODOLOGIA.....</b>	<b>18</b>
<b>3 COMPUTAÇÃO EM NUVEM.....</b>	<b>20</b>
<b>3.1 HISTÓRIA.....</b>	<b>21</b>
<b>3.2 MODELOS DE IMPLANTAÇÃO.....</b>	<b>24</b>
<b>2.2.1 Nuvem Pública .....</b>	<b>24</b>
<b>3.2.2 Nuvem Privada .....</b>	<b>26</b>
<b>3.2.3 Nuvem Comunitária .....</b>	<b>27</b>
<b>3.2.3 Nuvem Híbrida .....</b>	<b>27</b>
<b>3.3 PRINCÍPIOS DA COMPUTAÇÃO EM NUVEM .....</b>	<b>28</b>
<b>3.4 MODELOS DE SERVIÇOS .....</b>	<b>30</b>
<b>4 CONTAINER VS VIRTUALIZAÇÃO .....</b>	<b>32</b>
<b>4.1 LXC CONTAINERS .....</b>	<b>34</b>
<b>5. DOCKER .....</b>	<b>37</b>
<b>5.1 MOTIVOS PARA USAR O DOCKER.....</b>	<b>37</b>
<b>5.1.2 Instalação do Docker .....</b>	<b>39</b>
<b>5.2 ARQUIVOS DE CONFIGURAÇÃO .....</b>	<b>39</b>
<b>5.2.1 Docker-Compose .....</b>	<b>40</b>
<b>5.2.2 Docker File .....</b>	<b>42</b>
<b>5.3 DOCKER IMAGEM .....</b>	<b>44</b>
<b>5.4 DOCKERHUB .....</b>	<b>46</b>
<b>5.5 DOCKER CONTAINER.....</b>	<b>48</b>
<b>5.5.1 Software de Gerenciamento de Containers .....</b>	<b>51</b>
<b>5.6 DOCKER SWARM .....</b>	<b>54</b>
<b>5.7 PLAY WITH DOCKER.....</b>	<b>57</b>
<b>5.8 COMUNIDADE E EMPRESARIAL.....</b>	<b>59</b>
<b>5.8.1 Empresarial.....</b>	<b>60</b>
<b>6 BOAS PRÁTICAS DE CONSTRUÇÃO DA APLICAÇÃO (DOZE FATORES) .....</b>	<b>62</b>
<b>6.1 OS DOZE FATORES .....</b>	<b>62</b>
<b>7 SOFTWARES DE ORQUESTRAÇÃO .....</b>	<b>64</b>
<b>8 ESTUDO DE CASO.....</b>	<b>65</b>
<b>8.1 OBJETIVO .....</b>	<b>65</b>
<b>8.2 CENÁRIO ATUAL .....</b>	<b>65</b>
<b>8.3 DESCRIÇÃO DO PROJETO .....</b>	<b>65</b>
<b>8.4 ENVOLVIMENTO .....</b>	<b>66</b>
<b>8.4.1. Abrangência .....</b>	<b>66</b>
<b>8.5 RESTRIÇÕES .....</b>	<b>66</b>
<b>8.6 PROPOSTA DE SOLUÇÃO TECNOLÓGICA ESCOLHIDA .....</b>	<b>67</b>
<b>8.8 DIAGRAMA DE ATIVIDADES .....</b>	<b>67</b>
<b>8.9 REGRAS DE NEGÓCIO.....</b>	<b>69</b>
<b>8.10 INTERFACE VISUAL.....</b>	<b>70</b>

<b>9 INFRAESTRUTURA.....</b>	<b>75</b>
<b>10 CONCLUSÃO .....</b>	<b>77</b>
<b>11 TRABALHOS FUTUROS .....</b>	<b>78</b>
<b>12 REFERÊNCIAS .....</b>	<b>79</b>
<b>13 APÊNDICE .....</b>	<b>83</b>
Apêndice 13.1 – Dockerfile - X64 .....	83
Apêndice 13.2 – Docker-compose versão 2 - X64 .....	85
Apêndice 13.3 – Dockerfile – ARM.....	87
Apêndice 13.4 – Docker-compose versão 2 - ARM .....	89
Apêndice 13.5 – Docker-compose versão 3 - ARM .....	91

## INTRODUÇÃO

A computação em nuvem tem como enfoque proporcionar soluções com baixo custo de forma eficiente para o processamento, armazenamento e distribuições de montantes de dados. Atualmente, existem diversas definições e conceitos para a computação em nuvem. Neste estudo, irei utilizar a definição de (MELL AND GRANCE 2009) [1], onde de acordo com o mesmo, se pode definir computação em nuvem como sendo um modelo que provê acesso sob demanda a um conjunto de recursos computacionais, onde estes podem ser configurado de acordo com as necessidades, como CPU, armazenamento, memória e outros.

Estes recursos podem ser fornecidos e liberados de forma rápida, utilizando o mínimo de esforço de gerenciamento ou assistência do provedor da nuvem.

Com a revolução de dispositivos móveis e com a demanda crescente de aplicativos, dados, informações (dados e metadados), processamento e armazenamento; Os dados foram se tornando cada vez mais importantes, onde ao passar das épocas, foi-se tornando visível o avanço quanto a necessidade de dados e consequentemente, se foi percebendo um acumulo cada vez maior de informações.

A muito invisível para os usuários como seus aplicativos processam os seus recursos (metadados), dados e informações relevantes para a entrega de serviços; para engenheiros da computação e analistas é uma crescente preocupação de como poder crescente (escalar) cada vez mais provendo o máximo de recursos e dados frente a constante e crescente demanda dos clientes, sem ter o mínimo de *down-time* (tempo de falha) para novas implementações em produção (*deploy*).

Um novo conceito está surgindo e revolucionando a maneira de programar e suprir recursos para o ambiente das aplicações: programação em microserviços.

Este tipo de programação separa a aplicação em serviços, da qual a aplicação vai se encaixando como se fosse um bloco de caixas, sendo cada caixa um serviço específico. Estes serviços podem ser providos por módulos ligados a própria aplicação, ou módulos extensores de outras aplicações, que fazem interfaces por meio de *midlewares* (APIs).

Estes serviços não precisam estar no mesmo servidor que a aplicação, podem estar em outro servidor, em outro *Data Center* e até em outro continente.

Este conceito de serviços para servir aplicações está diretamente ligado ao novo paradigma programação e processamento em nuvem (programação distribuída), da qual cada serviço possa estar fora da aplicação e ser chamado conforme a sua demanda.

Devido a grande demanda de determinados serviços, essenciais a determinadas aplicações, estes precisam estar disponíveis quase que o tempo todo; porém grandes demandas podem ser um problema para os engenheiros de software, administradores de sistemas e a equipe de infraestrutura, sendo necessário aumentar a disponibilidade desse(s) serviço(s) subindo uma nova instância do mesmo;

O conceito de container visa resolver estes e outros problemas; se encaixa perfeitamente para a programação em microserviço, pois cada serviço fica isolado em um único container e este pode ser replicado (escalonado) conforme a demanda.

A minha linha de pesquisa não se fundamenta na aplicação em si, mais no conceito de serviço como infraestrutura e a escalabilidade que se pode ter para cada serviço, recurso, metadados, banco de dados e outros que possam estar servindo à aplicação. Irei fazer uma aplicação para fundamentar as teorias aplicadas nesta obra, porém a mesma será meramente aplicável ao conceito, não sendo o foco desta obra.

## 2 METODOLOGIA

Este trabalho utilizei como metodologia de pesquisa exploratória, visando aprendizado e conhecimento, utilizando em grande maioria fontes primárias e oficiais de conhecimento, como a documentação oficial do Docker e um livro lançado sobre o Docker para Desenvolvedores de Software, utilizei o Github desse livro, visto que o mesmo é de conteúdo livre e aberto.

Utilizarei como fonte secundária de conhecimento artigos e blogs sobre o tema e irei fazer uma entrevista, de conhecimento e utilização de plataforma de PaaS à uma equipe de desenvolvimento em uma empresa de internet de um grupo de mídias brasileiro. Não realizarei, quaisquer questionário e/ou perguntas prévias ao time de desenvolvimento, somente observarei o dia-a-dia da equipe e farei anotações sobre opiniões e conceitos que os mesmos já detenham.

Os resultados dessa pesquisa são de carácter qualitativos, me utilizo de uma prova de conceito, visando estabelecer um estudo de caso em uma infraestrutura própria.

Fiz a escolha deste tema, visto que obtive problemas no trabalho para poder escalar uma aplicação e me utilizei desse problema para aprendizado e conhecimentos sobre *Cloud* e serviços.

Este trabalho se dividirá além da introdução em outros capítulos: No capítulo 3, irei fazer uma fundamentação histórica e teórica sobre a computação em nuvem, modelos de implantação, suas vantagens e desvantagens, seus modelos de serviços, e informar sobre os tipos de nuvens existentes.

No capítulo 4, irei apresentar a informações sobre container e virtualização e o conceito do LXC Container.

No capítulo 5, irei falar sobre a plataforma a Open-source do Docker, informando como fazer a instalação, os arquivos de configuração utilizados no projeto e na plataforma, o conceito de imagem, repositório de imagens, container, softwares de gerenciamento de container utilizado e o conceito do Docker Swarm. Abordarei um projeto de um laboratório para aprendizado sobre o tema, o Docker, informarei sobre a comunidade do Docker e sobre a Docker empresarial.

No capítulo 6, irei falar sobre boas práticas adotadas em aplicações Web, utilizando o conceito de virtualização; são 12 fatores, boas práticas que são recomendadas de serem

seguidas para esses tipos de aplicações;

No capítulo 7, irei informar levemente sobre *PaaS* de orquestração, não irei me aprofundar muito sobre esse capítulo, visto que o mesmo pode ser muito amplo e fazer algumas comparações, sem fundamentos mais detalhados seria de carácter impreciso e ruim.

No capítulo 8, irei falar sobre o estudo de casos abordado no trabalho, farei da fundamentação teórica do modelo de visão do projeto, dando ênfases nos dados do software e regras de negócios da aplicação, o foco desta obra não é o software utilizado no estudo de caso, faço uso do mesmo somente para poder fazer comprovações sobre o estudo do Docker nesta obra.

No capítulo 9, irei falar sobre infraestrutura de cluster utilizada, como fundamentação desta obra e a aplicabilidade deste modelo.

No capítulo 10, irei fazer a conclusão do meu trabalho; Abordando meu ponto de vista e críticas e soluções relativas à este novo modelo e paradigma de programação.

No capítulo 11 irei informar o meu ponto de vista sobre trabalhos futuros, se por ventura que esta obra possa ser continuada.

No capítulo 12 são as referências utilizadas como base de aprendizado e consulta desta obra.

No capítulo 13 irei anexar os arquivos de configurações das arquiteturas utilizadas nesta obra, para conhecimento e continuação, se houver.

### 3 COMPUTAÇÃO EM NUVEM

O termo de computação em nuvem se refere à entrega de recursos computacionais através da Web ou de uma rede própria.

De acordo com o Instituto Nacional de Padrões e Tecnologia do Departamento de Comércio Norte-Americano:

Computação em nuvem é um modelo para permitir acesso ubíquo, conveniente e sob demanda via rede a um agrupamento compartilhado e configurável de recursos computacionais (por exemplo, redes, servidores, equipamentos de armazenamento, aplicações e serviços), que pode ser rapidamente fornecido e liberado com esforços mínimos de gerenciamento ou interação com o provedor de serviços. (NIST, 2015) [1].

Ao invés de manter servidores e infraestrutura especializada para processamento de dados, *storage* de arquivos e outros serviços, utiliza-se serviços online (pode ser de outras empresas) para tal fim. Porém ao se expor a esse tipo de serviço, torna-se necessário ter algum tipo de questionamento e expertise para se manter a segurança, a confiabilidade e a privacidade desses dados.



Figura 1: Cloud Fonte: <http://www.synergixtech.com/wp-content/uploads/2016/09/Cloud-Computing-Benefits.png>

### 3.1 HISTÓRIA

A principal funcionalidade da computação em nuvem é a utilização de recursos computacionais por meio da web, e a idéia não é tão recente, a ideia já existia em 1960, com Joseph Carl Robnett Licklider [2].

Joseph Carl foi um dos desenvolvedores da *ARPANET (Advanced Research Projects Agency Network)*, o antecessor direto da internet, que tinha o objetivo de interligar as bases militares e os departamentos de pesquisa do governo americano.

Nesse período, Joseph já imaginava uma rede de computadores intergaláctica em que todos estariam conectados acessando programas e dados de qualquer lugar.



Figura 2: Joseph Carl  
Fonte: <http://www.psynergie.com/psychologie-internet/photo-joseph-licklider.jpg>

Na década de 1960, John McCarthy, um importante americano pesquisador da área da informática e também um dos pioneiros da inteligência artificial, propôs a ideia de que a computação deveria ser organizada na forma de um serviço de utilidade pública, assim como

os serviços de água e energia, em que os usuários só pagam pelo que usam, sendo precursor da idéia de *PaaS*.

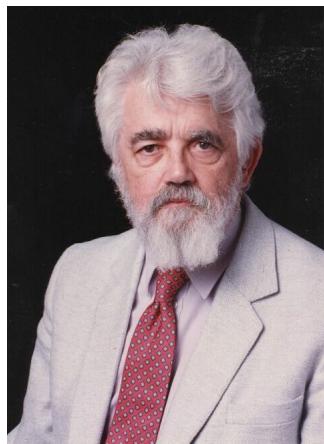


Figura 3: John McCarthy  
Fonte: <http://www-formal.stanford.edu/jmc/jmccolor.jpg>

Mesmo com a existência dessas ideias há tanto tempo, o termo computação em nuvem só veio a ser mencionado em 1997, numa palestra acadêmica do professor de Sistemas da Informação Ramnath Chellappa, e só foi desenvolvida no ano de 1999 com o surgimento da Salesforce.com, primeira empresa a disponibilizar aplicações pela internet.



Figura 4: Ramnath Chellappa  
Fonte: <http://goizueta.emory.edu/profiles/images/portrait/chellapa.jpg>

A partir do sucesso dessa empresa, outras grandes começaram a investir na área, como a Amazon®, a Google®, a IBM® e a Microsoft®.

O que conhecemos hoje como computação em nuvem, nasceu com os sistemas distribuídos, caracterizados por serem um conjunto de unidades de processamento independentes, que, por meio da troca de comunicação e gerenciamento de sincronização, pode processar uma aplicação em diferentes localidades, de forma transparente para o usuário, ou seja, o usuário da aplicação vê apenas o todo. A computação em nuvem vai além disso, trata-se de um formato de computação a partir do qual aplicativos, serviços, dados e recursos de *TI* são disponibilizados aos usuários como serviço, por meio da internet e/ou de uma rede própria.

Não é mais necessário, para algumas empresas, ter supercomputadores, pois o poder de processamento e os dados ficam nas nuvens. Só precisamos de dispositivos que nos dêem acesso a esses recursos. Dispositivos estes que consequentemente são mais baratos e possuem uma maior portabilidade e flexibilidade, como smartphones, tablets e notebooks.

### 3.2 MODELOS DE IMPLANTAÇÃO



Figura 5: Modelos de Implementação Fonte: <http://www.vividdynamics.com/wp-content/uploads/2013/12/cloud-hosting.jpg>

É possível implantar soluções utilizando computação em nuvem de maneiras diferentes, dependendo de fatores, como:

- Requisitos de segurança;
- Hospedagem dos serviços;
- Capacidade de customização;
- Nível de acesso;
- Gerenciamento de serviços;

Existem modelos principais que são determinados pelo nível de acesso: Nuvem pública, Nuvem privada, Nuvem Comunitária e Nuvem híbrida.

#### 3.2.1 Nuvem Pública



Figura 6: Nuvem pública Fonte: [http://www.ximedica.info/images/uploads/the\\_cloud-resized-600.jpg](http://www.ximedica.info/images/uploads/the_cloud-resized-600.jpg)

Uma infraestrutura de nuvem pública é disponibilizada para o público geral e é de propriedade de um provedor de serviços de web (terceirizado). Em uma nuvem pública, os recursos de computação são disponibilizados dinamicamente através do provedor e são fornecidos para seus clientes (rede própria utilizando *VPNs* tendo como base a infraestrutura da web); A cobrança feita pelo provedor é feita pela quantidade de recursos que o cliente utiliza (isto pode variar em contrato).

Esse modelo possui boa relação custo/benefício para o cliente, uma vez que oferece a flexibilidade de disponibilizar apenas os recursos necessários e entregar todos os serviços com certa consistência de disponibilidade, resiliência, segurança e facilidade de gerenciamento. Como este modelo é baseado na web e possui infraestrutura e meios de acesso administrados pelo provedor de acesso o cliente precisa aceitar o controle reduzido e o monitoramento feito do provedor, além de confiar na governança e segurança da infraestrutura servida pelo provedor.

### 3.2.2 Nuvem Privada

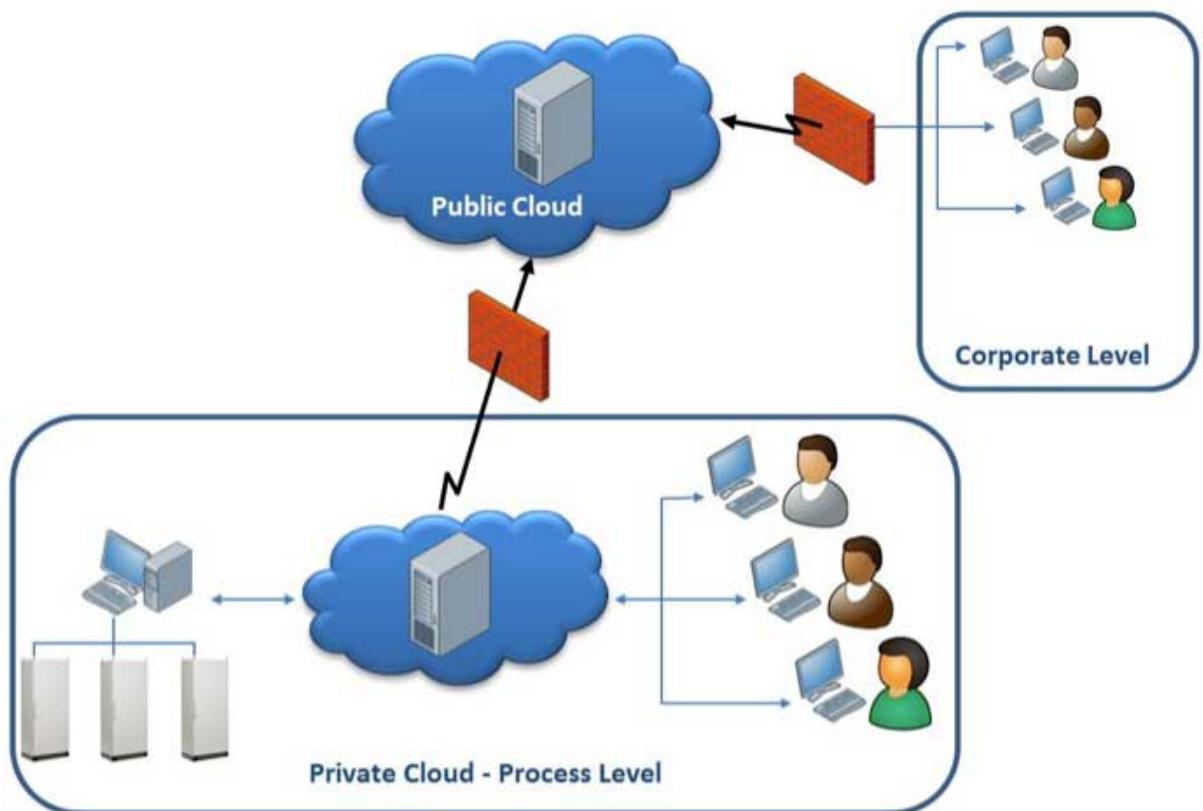


Figura 7: Nuvem Privada Fonte: [https://web-material3.yokogawa.com/image\\_8952.jpg](https://web-material3.yokogawa.com/image_8952.jpg)

Uma infraestrutura de nuvem privada é operada apenas para um único cliente, o provedor dedica serviços específicos para aquele cliente. O cliente especifica, arquiteta e controla toda a gama de recursos computacionais que o provedor fornece. Um motivo comum que leva os clientes a procurarem um serviço de nuvem privado é a capacidade de controlar e garantir seus próprios padrões de segurança dos dados.

Um cliente normalmente fará uso de uma nuvem pública utilizando conexões através de links privados e esses recursos apenas serão compartilhados internamente. Como os recursos não são compartilhados entre várias organizações, o cliente, que contratou o serviço, paga o valor total pelos recursos da nuvem, independente da quantidade que foi utilizada.

Sendo assim, a organização contratante pode realocar os recursos para subáreas da mesma empresa conforme suas necessidades.

### 3.2.3 Nuvem Comunitária

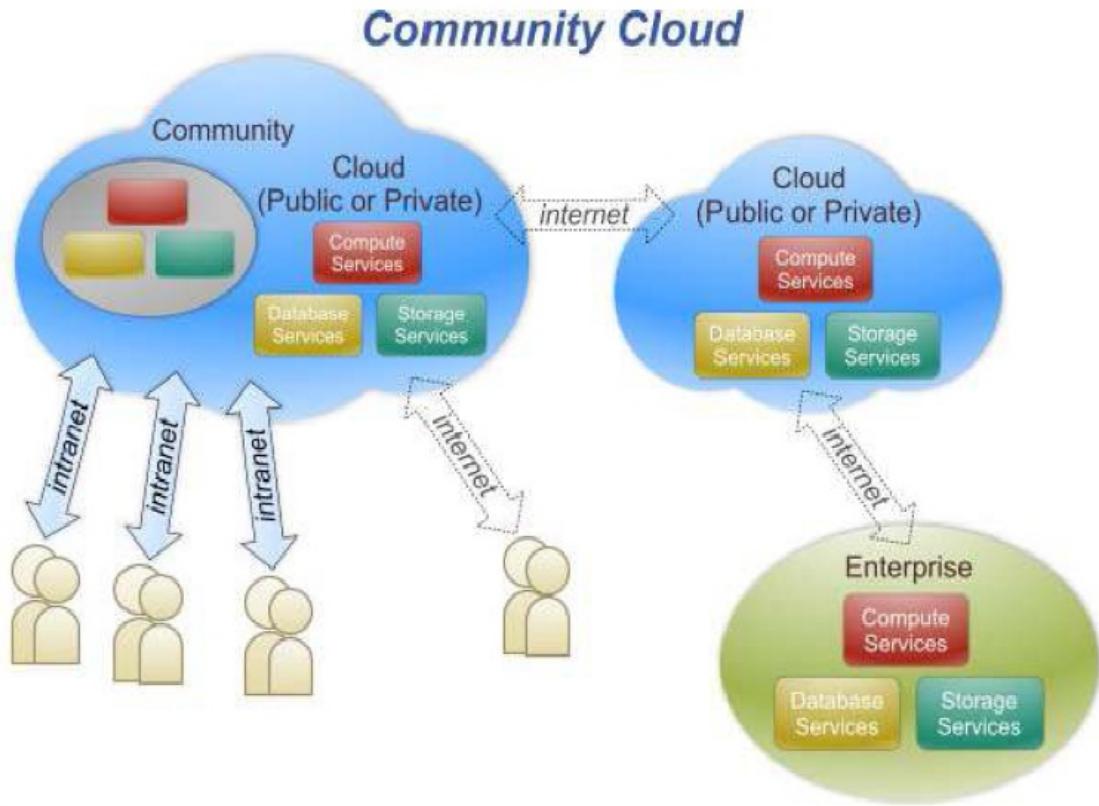


Figura 8: Nuvem Comunitária Fonte: [https://lh4.googleusercontent.com/NWf67CzmfXLVsJ60ZRyC-eXUcdAt3ITRZcgedyN4dBbGU0BOWdCSNdzuqz9DxZ4fHNC6GLnUlreeoRX\\_8c07l61YMMoY3zxKJvMkbFfx92vjjDYLm1ai2STm0h4XQfA](https://lh4.googleusercontent.com/NWf67CzmfXLVsJ60ZRyC-eXUcdAt3ITRZcgedyN4dBbGU0BOWdCSNdzuqz9DxZ4fHNC6GLnUlreeoRX_8c07l61YMMoY3zxKJvMkbFfx92vjjDYLm1ai2STm0h4XQfA)

Uma infraestrutura de nuvem comunitária é contratada por um grupo de organizações em conjunto ou programas que compartilham necessidades específicas, como segurança e aspectos legais. O controle da nuvem pode ser feito pelo cliente ou pelo provedor, de acordo com o que foi combinado no contrato.

Quando organizações possuem o mesmo conjunto de requisitos e clientes, a nuvem comunitária permite a eles combinarem ferramentas e compartilharem recursos computacionais, dados e capacidades. Ao eliminar a duplicidade de sistemas similares, as organizações podem economizar dinheiro e alocar seus recursos de maneira mais eficiente.

### 3.2.3 Nuvem Híbrida

## Deployment Models

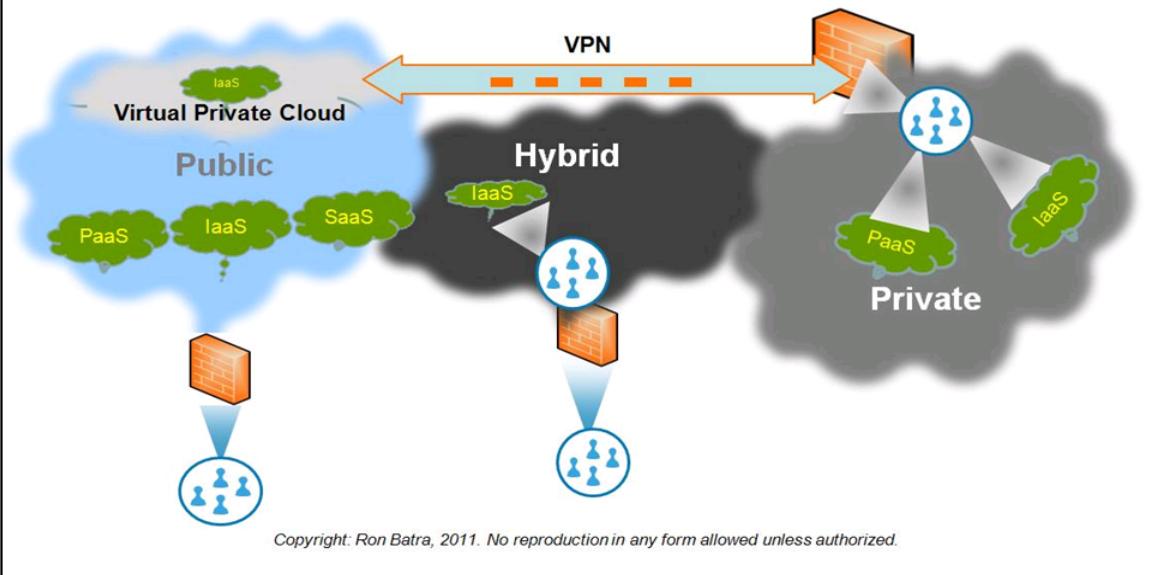


Figura 9: Modelos de Implementação Fonte:  
<https://puserscontentstorage.blob.core.windows.net/userimages/de1cc483-bb71-4170-bd25-0c04f167acf5/c9851e30-da98-4765-92bb-d33ca089ff49image32.png>

Uma infraestrutura de nuvem híbrida abrange duas ou mais nuvens, podendo estas serem nuvens públicas, comunitárias ou privadas, com o conjunto dos serviços que são hospedados internamente ou externamente.

Os clientes geralmente não se limitam a um único modelo de implantação, mas sim preferem incorporar serviços de nuvem diferentes e sobrepostos para atingir seus requisitos específicos. Modelos de implantação híbridos são complexos e requerem um planejamento específico para serem executados e gerenciados especialmente quando é necessária a comunicação entre dois tipos diferentes de implantações em nuvem.

### 3.3 PRINCÍPIOS DA COMPUTAÇÃO EM NUVEM

Segundo NIST (2011) [1], um modelo de Computação em Nuvem deve apresentar algumas características essenciais:

- Autoatendimento sob demanda: o usuário pode usufruir das funcionalidades computacionais sem a necessidade da interação humana com o provedor de serviço, ou seja, o provedor identifica as necessidades do usuário, podendo assim automaticamente reconfigurar todo hardware e software, e essas modificações devem ser apresentadas ao usuário de forma transparente.

- Amplo acesso a serviços de rede: os recursos computacionais são acessados através da internet, que são acessados por mecanismos padronizados, que pode ser um navegador simples, que use poucos recursos computacionais, sem a necessidade de o usuário modificar o ambiente de trabalho de seu dispositivo, como por exemplo, linguagem de programação e sistema operacional.
- Pool de recursos: os recursos computacionais (físicos ou virtuais) do provedor são divididos em pools para que possam atender a múltiplos usuários simultaneamente. Esses recursos são alocados e realocados dinamicamente, de acordo com a demanda dos usuários. Os usuários por sua vez não precisam saber a localização física dos recursos computacionais, essas informações podem ser proporcionadas de maneira de alta abstração podendo apenas ser informados o país, estado ou centro de dados.
- Elasticidade rápida: as funcionalidades computacionais devem ser rápidas e elásticas, assim como rapidamente liberadas, podendo em alguns casos serem liberadas automaticamente caso haja necessidade devido a demanda. O usuário deve ter a impressão de ter recursos ilimitados que podem ser comprados ou adquiridos em qualquer quantidade e a qualquer momento. A elasticidade deve ter três componentes: escalabilidade linear, utilização on-demand e pagamento por unidades consumidas de um recurso. Outro recurso que pode auxiliar nesse processo é a virtualização que pode criar várias instâncias de recursos requisitados usando apenas um recurso físico. A virtualização também torna possível abstrair características físicas de uma plataforma computacional, emulando vários ambientes que podem ser independentes ou não.
- Serviços mensuráveis: os sistemas em nuvem automaticamente controlam e monitoram os recursos necessários para cada tipo de serviço, tais como armazenamento, processamento e largura de banda. Esse recurso deve ser monitorado e controlado de forma transparente tanto para o provedor de serviço quanto para o usuário.

Além dessas características, algumas outras não tão essenciais em um ambiente de computação em nuvem, mas que definem para o bom serviço às aplicações estão hospedadas e o bom relacionamento com o cliente, são essas:

- Tolerância a falhas: O provedor de serviço tem que ser totalmente redundante em sua infraestrutura e possibilitar alta disponibilidade de dados para os seus clientes, de modo que se houver alguma falha ou problema em sua estrutura de

rede, servidor, processamento, armazenamento e outros, os dados e aplicações dos clientes possam ser realocados para outras máquinas (servidores), datacenters, cluster e até continentes sem que o cliente perceba que tal falha ocorreu.

- Níveis de Qualidade de Serviço de *SLA*: Esta questão está diretamente ligada com a tolerância a falhas, da qual em último caso de a falha de fato ocorrer que o provedor de serviço possa dar o mínimo de assistências aos seus usuários e que o mesmo, possam ter alguma garantia de que os seus serviços, possam ser restabelecidos dentro do tempo referenciado em contrato.

### 3.4 MODELOS DE SERVIÇOS

A idéia de Computação em Nuvem é composta por modelos de serviços, esses modelos são pagos conforme a necessidade e o uso dos mesmos (*pay-per-use*), dando ao cliente a possibilidade de usar mais ou menos recursos de acordo com sua necessidade. Os modelos de serviços são os seguintes:

- Software como Serviço (*SaaS*): um aplicativo pode ser utilizado por uma grande quantidade de usuários simultaneamente. Esse tipo de serviço é disponibilizado por provedores e acessado pelos usuários através de aplicações como o navegador. Todo o controle e gerenciamento da rede, sistemas operacionais, armazenamento e possíveis manutenções será de responsabilidade do provedor de serviço [AULBACH, 2009][ 4].
- Plataforma como Serviço (*PaaS*): é a disponibilização de plataformas de desenvolvimento que facilitam a implantação de aplicações assim como o gerenciamento do hardware subjacente e das camadas de software. O usuário não tem controle sobre a rede, sistemas operacionais ou armazenamento, mas poderá controlar a aplicação implementada na nuvem. A linguagem de programação bem como o ambiente de desenvolvimento é fornecida pelo provedor (NOGUEIRA, 2010) [3].
- Infraestrutura como Serviço (*IaaS*): consiste no fornecimento de infraestrutura de processamento, armazenamento, redes, entre outros. Este serviço, assim como os demais, tem seus recursos – neste caso a infraestrutura – compartilhados com diversos usuários simultaneamente. Isso se torna possível através do processo de virtualização,

no qual o usuário terá controle sobre máquinas virtuais, armazenamento, aplicativos instalados e possivelmente um controle limitado sobre os recursos de rede (VERAS, 2012) [6].

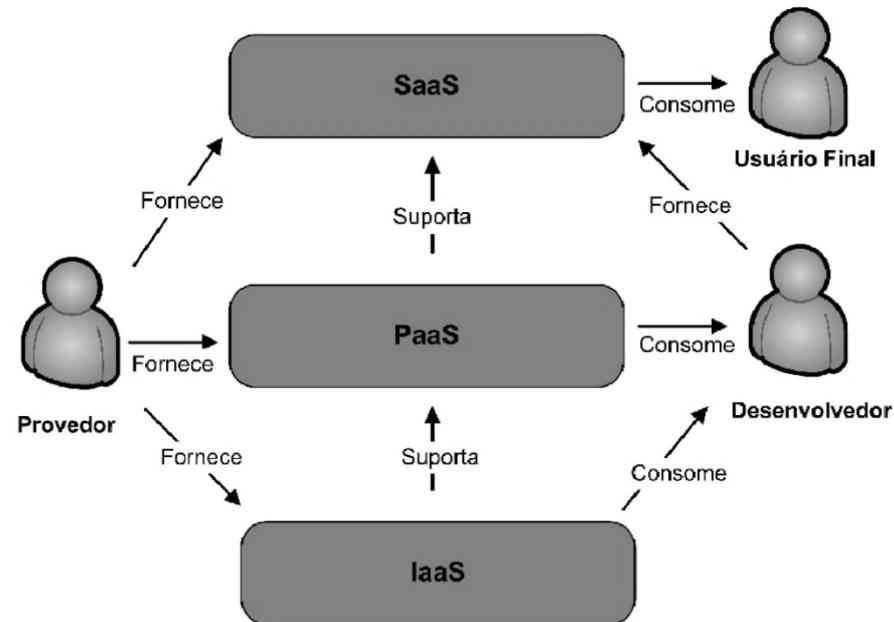


Figura 10 - Modelos de Serviço  
Fonte:  
<http://ftp.unipar.br/~seinpar/2013/artigos/Rogerio%20Schueroff%20Vandresen.pdf>

## 4 CONTAINER VS VIRTUALIZAÇÃO

Os sistemas de virtualização passaram por algumas mudanças ao longo do tempo até chegarem aos sistemas atuais, em primeira fase eram servidores em grande escala que ocupavam grande parte de uma sala, sem virtualização e que rodavam somente uma aplicação, não usando nem metade do seu poder de processamento e com grandes gastos de energia; Se necessário ampliação havia custos de compra de hardware e de espaços, energia, gerenciamento centralizado das máquinas, segurança entre outros.

Com o surgimento da virtualização houve a inserção de um hypervisor no sistema operacional, que consiste em um gerenciador para a virtualização.

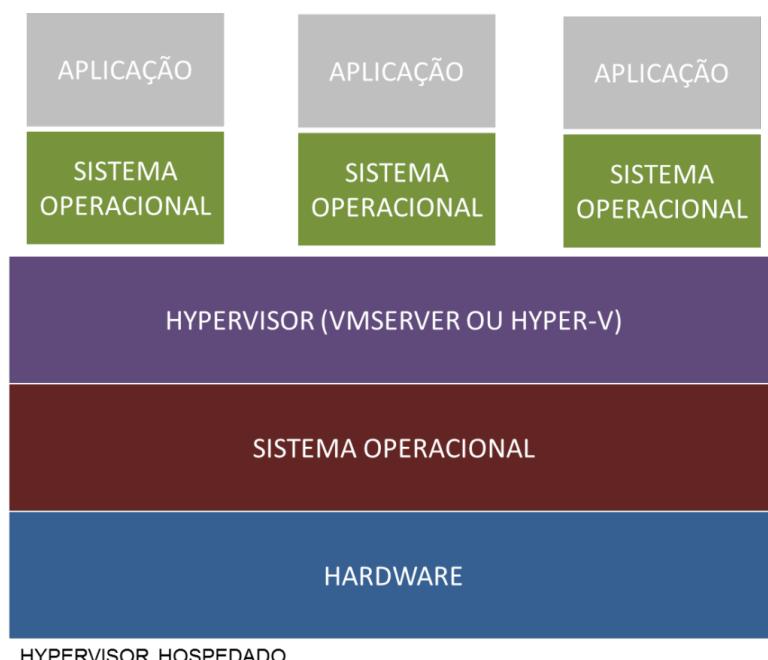


Figura 11 Hypervisor Hospedado Fonte: <http://3way.com.br/saiba-como-a-virtualizacao-por-container-mudou-a-infraestrutura-de-ti/>

A evolução deste modelo consiste em não usar mais a camada do sistema operacional, e sim que o próprio hypervisor é que faz a gestão em cima da camada do hardware, se tornando um sistema operacional de gerenciamento.

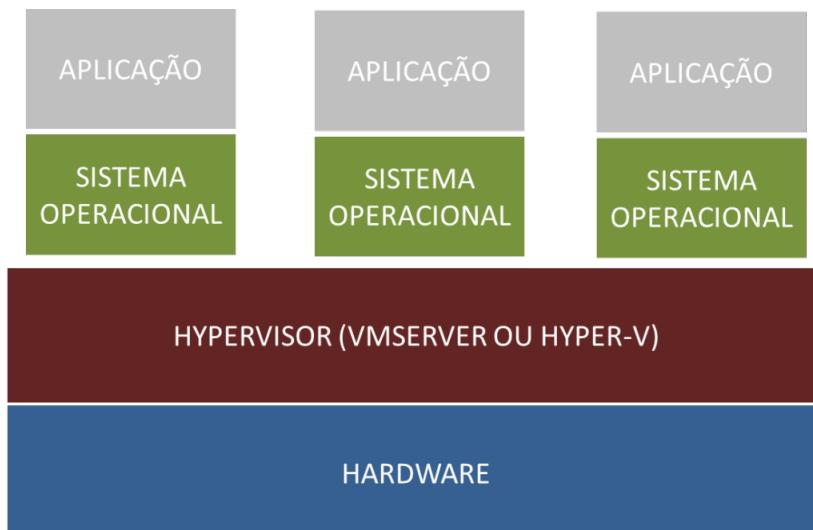


Figura 12 - Evolução do Hypervisor Fonte: <http://3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti.png>

O modelo de containers elimina a camada do sistema operacional que existia para cada máquina virtualizada e o hypervisor para gerenciar as instâncias virtualizadas. A virtualização pelo container utiliza o kernel do sistema operacional nativo da máquina, geralmente sendo o Linux, mas hoje já é possível ser feito em outros sistemas operacionais.

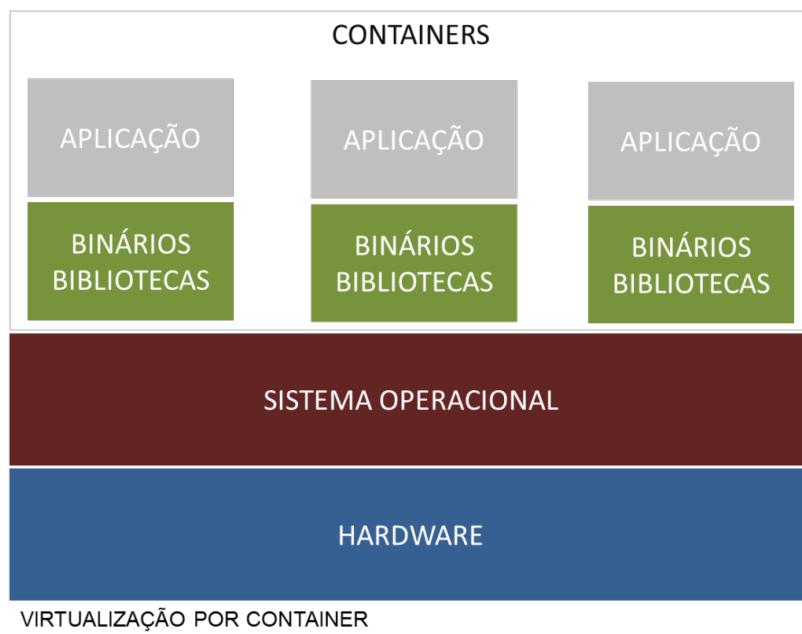


Figura 13 - Virtualização por Container Fonte: <http://3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti.png>

O kernel do sistema é responsável por fornecer as bibliotecas e os binários necessários para que o container possa rodar a aplicação de forma necessária.

Basicamente há uma abstração do nível de sistema operacional, pois as configurações ficam no container e o mesmo possui um alto nível de provisionamento, podendo ser replicado sem menores problemas, simplificando muito a implementação em diferentes máquinas e ambientes de homologação (QA, STAGING, PROD); o container pode ser baixado em qualquer host, que possui um cliente do Docker instalado, utilizando o Dockerhub (repositório do Docker).

Esse dinamismo foi um dos alicerces da computação em nuvem, permitindo replicações de recursos, aplicações e escalabilidade a níveis nunca vistos antes.

## 4.1 LXC CONTAINERS

Conforme citação do Rogério dos Anjos (em Linux, Novidades):

“O LXC (Linux Container) é um sistema leve de virtualização que usa múltiplos containers de forma isolada no kernel Linux. Ele cria um ambiente muito próximo de um sistema Linux sem precisar instalar um kernel separado. Com o LXC é possível criar processos separados para usuários, espaço em disco, memória, CPU, rede e muito mais”.

[9]

Conforme citação de Cristiano Diedrich:

“O projeto do LXC, trazia as seguintes fases: LXC, chroot com esteróides. O objetivo do projeto era ser uma alternativa a já consolidada tecnologia de chroot, sendo um meio termo entre máquina virtual e chroot, possibilitando a criação de um ambiente mais próximo possível de uma instalação Linux sem a necessidade de um kernel separado.”[12]

O container tem a característica de isolar os recursos do sistema dos recursos da aplicação, criando assim um ambiente isolado; o mesmo poder consumir recursos do sistema como: namespace, chroot, cgroups entre outros. Funciona com se cada container fosse uma máquina virtual completa, podendo ter inclusive os mesmos problemas de uma máquina virtual, como criação de usuários e permissões de escritas em diretórios e etc.

O resultado é uma máquina virtual sem a camada do hypervisor, isolada e com controle de recursos.

Conforme citação do Rogério dos Anjos (em Linux, Novidades):

“Os containers fornecem um ambiente mais próximo possível de um sistema operacional do que você conseguiria de uma máquina virtual, mas sem a sobrecarga da execução separada do kernel e da simulação de hardware do sistema”[10].

Os benefícios da utilização do LXC container, viabilizou grandes benefícios para a virtualização e para a computação em nuvem, como:

- Provisionamento: Sendo praticamente rápido e simples inicializar um container, quase que instantaneamente, demorando alguns minutos, bem mais rápido do que um provisionamento por uma máquina virtual.
- Escalabilidade: O provisionamento dinâmico do LXC container, permite criar instâncias conforme a demanda, promovendo disponibilidade de serviço.
- Custo: Tendo em vista que para a virtualização de máquina, na maioria das vezes é feito sobre plataformas de virtualização e isto pode implicar em custos adicionais para empresas que podem precisar de suporte especializado. Isto implica em custos de licenças e suporte. O projeto do LXC container e algumas plataformas de gerenciamento e deploy de containers são de caráter open-source.

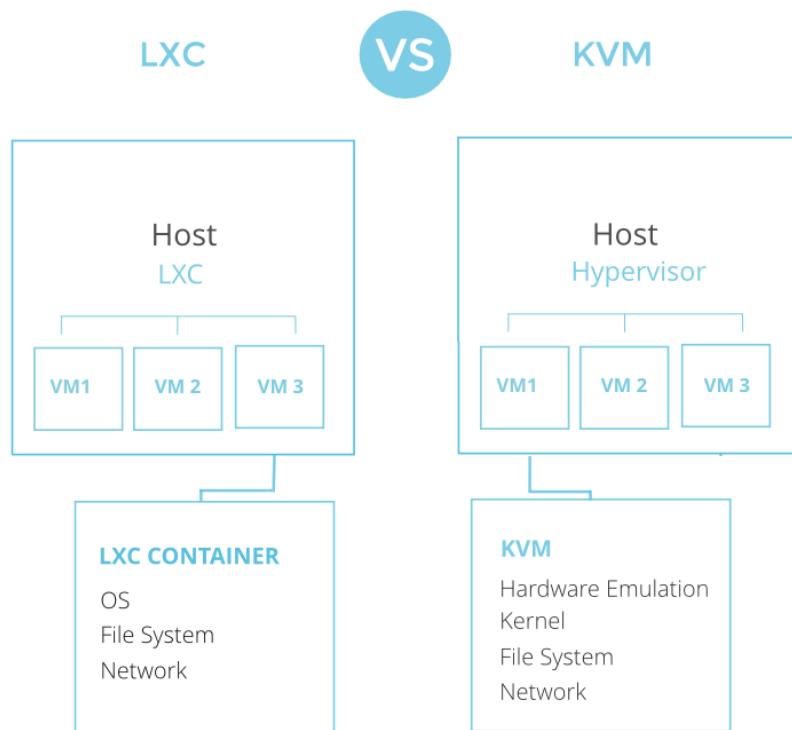


Figura 14 - LXC vs KVM Fonte: <http://3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti-part2/.png>

A tecnologia de containers é a base do Docker, que será mostrado à seguir; Durante este trabalho irei falar sobre alguns benefícios de se usar o docker e a tecnologia de containers; porém de acordo com Rafael Benevides – Diretor de Experiência de Desenvolvimento da Red Hat[13] - existem alguns passos à serem seguidos para execução com containers:

- Primeiro: Containers são imutáveis - O Sistema Operacional, versões de bibliotecas, configurações, pastas e a aplicação são empacotados dentro do container. Isto garante que a mesma imagem que foi testada em QA, irá reagir no ambiente de produção com o mesmo comportamento.
- Segundo: Containers são leves – A memória dentro do container é pequena. O container apenas alocará memória para o processo principal
- Terceiro: Containers são rápidos – É necessário instanciar um container de forma rápida, assim como um típico processo Linux inicia. Instaciado em minutos, é necessário iniciar um novo container em segundos.

## 5. DOCKER

De acordo com a documentação oficial do Docker:

O Docker é uma plataforma aberta de desenvolvimento, entrega e execução de aplicações. Docker permite separar sua aplicação da sua infraestrutura para fazer entregas de forma rápida.

Tradução nossa, “Com o Docker, é possível gerenciar a sua infraestrutura da mesma forma que se gerencia sua aplicação. Mais com a vantagem da metodologia do Docker de carregar, testar e entregar o código de forma rápida, você pode reduzir significativamente o tempo entre escrever o código e rodá-lo em produção [14].

O Docker possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um container, e a partir desse momento o ambiente inteiro torna-se portável para qualquer outro *host* que contenha o cliente do Docker instalado.

Isso reduz drasticamente o tempo de *deploy* de alguma infraestrutura ou até mesmo aplicação, pois não há necessidade de ajustes de ambiente para o correto funcionamento do serviço, o ambiente é sempre o mesmo, configure-o uma vez e replique-o quantas vezes quiser. Este ambiente poderá estar disponível no *DockerHub* que é o repositório de imagens do Docker, irei falar mais adiante sobre este ambiente em maiores detalhes.

Muito utilizado para testes de homologação de aplicações, pois é possível criar um ambiente de *QA* igual ao ambiente de *PROD* de forma que possa ser homologado com segurança e governança.

### 5.1 MOTIVOS PARA USAR O DOCKER

Existem algumas refutações para poder usar o Docker, principalmente sobre entendimento e a aplicabilidade desta tecnologia nas empresas e adoção dos desenvolvedores.

De acordo com o livro Docker para desenvolvedores:

Vale frisar que o Docker não é uma “bala de prata” - ele não se propõe a resolver todos problemas, muito menos ser a solução única para as mais variadas situações. [15]

O Docker trás alguns benefícios para determinados cenários de aplicações:

- Abstração do host separada da aplicação
  - Como já informado anteriormente há uma separação do host e da aplicação. Utilizando o conceito do LXC container e algumas outras ferramentas, como o Docker-compose, irei explicitá-lo em outro capítulo.
- Fácil Escalabilidade
  - Fácil replicação de algum recurso/serviços com a duplicação do container, utilizando scripts e/ou plataformas de automatização.
- Gerenciamento simples de dependências e versionamento das aplicações
  - Como a aplicação fica isolada, a mesma pode possuir todas as suas dependências (variáveis de ambiente, bibliotecas e outras dependências) no próprio container.
- Ambientes de execuções leves e isolados
  - Cada aplicação ou recurso fica isolado, somente com o que é necessário à sua execução.
- Camadas compartilhadas
  - Compartilhamento de camadas comuns a nível de Sistema Operacional, como bibliotecas, Kernel e etc.

### **Problemas de usar o Docker:**

- Necessário adaptar a aplicação ao Docker.
- Necessário conhecimento, e/ou experiência com o uso de container/imagem.
- Curva de aprendizagem demorada e que precisa de experiência do usuário.
- Pouca documentação para conceitos/problemas envolvendo à aplicação.
- Erros na criação do container/imagem são de difícil compreensão, assimilação e entendimento. Necessária experiência do usuário.
- Necessário conhecimento prévios de Linux.

Principal problema do Docker, é que a equipe de desenvolvimento tem que produzir a aplicação já pensando na forma de execução da mesma no Docker. Em meu estudo de caso, precisei fazer alterações significativas na aplicação para que a mesma pudesse ser executada no Docker.

Usuário leigos e/ou sem um bom conhecimento de Linux terão extrema dificuldade de utilização da plataforma, pois alguns erros não são claros e requerem bastante conhecimento e motivação do usuário para continuar a *debugar*. Em meu estudo de caso varias vezes a minha aplicação não levantava, ficava “up” e somente obtive um “*stack trace de exit=1*”; pois um container dependente não era iniciado de forma correta e o container minha aplicação me informava esse stack trace de erro.

### 5.1.2 Instalação do Docker

#### Instalando no GNU/Linux

A instalação do Docker é bem simples e de uma forma bem genérica, segue os comandos usados GNU/Linux

#### Docker engine no GNU/Linux

Para instalar o Docker engine é simples. No terminal do GNU/Linux é necessário se tornar usuário root:

- sudo su - root

Execute o comando abaixo:

- wget -qO- https://get.docker.com/ | sh

#### Instalando no MacOS

A instalação do Docker no MacOS pode ser realizada através do brew cask, com o comando abaixo:

- brew cask install docker

Ou é possível instalar o cliente pelo próprio site do Docker

## 5.2 ARQUIVOS DE CONFIGURAÇÃO

Os arquivos de configuração do Docker, são arquivos que vão repassar para a *API* do Docker a forma de o mesmo irá prosseguir, criando as imagens, containers e serviços.

Esses arquivos são úteis para criar configurações personalizadas e gerenciar a forma que os serviços vão ser executados pelo *Daemon* do Docker.

Na documentação oficial do Docker é possível ver uma relação de boas práticas [17] para a melhor construção desse arquivo.

Possuo dois arquivos de configurações do Docker-compose e Dockerfile adaptados para as arquiteturas X64 e ARM. Foi necessário fazer essas adaptações devido a diferença de arquiteturas de processadores que executarão os códigos.

Possuo duas arquiteturas, pois fiz a implementação do estudo de caso em um cluster ARM, com Raspberry Pi 3, irei falar mais adiante sobre o mesmo.

Arquivos das Arquiteturas:

- X64:
  - Docker file: Apêndice 13.1
  - Docker-compose: Apêndice 13.2
- ARM:
  - Docker file: Apêndice 13.3
  - Docker-compose versao 2 : Apêndice 13.4
  - Docker-compose versao 3 : Apêndice 13.5

### 5.2.1 Docker-Compose

Existe uma ferramenta do Docker que é o Docker-compose, da qual é possível escrever um único arquivo em formato “.yml” e o mesmo criará os containers em formato de serviços. Esses serviços serão executados na ordem que forem escritos no arquivo e o mesmo irá criar os containers de acordo com as configurações descritas nesse arquivo “.yml”

De acordo com a documentação oficial do Docker-compose:

Tradução nossa, “Compose é uma ferramenta para definição e execução de aplicações complexas com Docker. Com o Docker-compose, é possível definir múltiplos containers em um único arquivo. Então levante a sua aplicação com um único comando que faça o que for preciso para executá-la.”

Toda a escrita do arquivo do Docker-compose é no formato “.yml” uma linguagem bem próxima da linguagem natural e que pode ser facilmente compreendida e interpretada por uma simplicidade e quase nenhum conhecimento prévio específico de programação .

O Docker-compose utiliza o conceito de execução por serviços, da qual cada container é executado como um serviço e o mesmo pode estar dependente e vinculado a outro serviço (s) para poder executar.

Dependente significa que o mesmo depende de outro serviço para poder executar, um exemplo: o *Redis*. A aplicação que está sendo mostrada neste estudo de caso, depende do redis para poder executar; isso significa que o container do redis tem que ser construído “build” e executado antes do container da aplicação. Essa ordem é de suma importância e está referenciada na ordem de execução dos serviços escritos no arquivo “.yml”.

Estar vinculado significa que um container estará prestando serviço para outro container. Ambos são independentes e podem ser construídos (build) e executados em separado. Por exemplo o banco de dados em MySQL ele presta serviços para a aplicação, a mesma consegue ser construída (build) e ficar “up” sem o banco; não vai ter sua completa utilização e execução sem o mesmo, mas é possível.

Todos os serviços precisam estar vinculados no arquivo “.yml”, fazendo referência à imagem que foi criada pelo Dockerfile. A imagem é construída antes da execução do container que faz referência, os containers são criados seguindo a ordem que está descrita no arquivo “yml”.

Ordem de execução:

- Docker File
- Imagem
- Docker-compose
- Containers

### **Comandos do Docker-compose utilizados:**

- Build — Descrição do comando no terminal – Criar e/ou recriar serviços.

Serviços são tageados com o nome da aplicação seguido do nome do serviço por underline.

- Up — Descrição do comando no terminal – Criar, re(criar), iniciar o container de serviço. Se o container não existir será executado o build. Se o container já existir o mesmo será parado e recriado, preservando os volumes.

### 5.2.2 Docker File

De acordo com a documentação oficial do Docker:

Tradução nossa, “O Dockerfile é um documento de texto que contém comandos que normalmente serão executados manualmente no build da imagem. O Docker pode fazer o build automaticamente da imagem, lendo as instruções no arquivo do Dockerfile.” [17]

A escrita do arquivo do Dockerfile é em formato Shell-Script, não sendo tão próxima da linguagem natural, como o arquivo do Docker-compose, é de compreensão difícil, por se tratar de uma linguagem mais baixo nível e não comumente utilizada. Necessitando de conhecimentos prévios de computação para execução de tal.

Administradores de Sistemas e equipes de Infraestrutura, acostumados a fazer scripts para fazer alterações em lote para Sistemas Operacionais Linux, possuem maior compreensão deste tipo de linguagem e dos comandos de Sistema Operacional utilizados.

Essa técnica possui algumas aplicações:

- Isolamento do sistema operacional
  - É possível utilizar um sistema operacional específico para a imagem, pode até ser diferente do sistema operacional nativo do host, da qual o Docker está sendo executado — Em versões mais antigas era necessário a instalação do Boot2docker — que faria essa camada de abstração do Docker ao Sistema Operacional. Porém já foi corrigido em novas versões do Docker; não sendo mais necessário.
- Controle de arquivos
  - Permite fazer um controle de diretórios e arquivos que serão interessantes e/ou necessários à aplicação acessar para manter a execução. Como por exemplo diretório de arquivos utilizados pela própria aplicação.
- Controle de Dependências

- É possível fazer a instalação específica de dependências (bibliotecas) necessárias à execução da aplicação.

Instalações que permanecerão para a imagem também podem ser inseridas no momento de construção do Docker file; Em meu estudo de caso para a arquitetura ARM eu faço a instalação do RVM, Ruby e bundler das dependências do projeto nesse arquivo diretamente. Estas serão necessárias à execução para esta arquitetura; visto que não consegui uma imagem funcional do Ruby, RVM e bundler para esta arquitetura. Irei aprofundar informações da imagem mais adiante.

Este arquivo é de suma importância para a arquitetura do Docker, pois será através dele que a imagem será criada e os containers terão como base para a sua execução; quaisquer esquecimento e/ou faltas de instalação de bibliotecas e dependências, implicarão na remoção das imagens e dos containers associados e recriação.

Durante os meus testes para o estudo de caso, tive extrema dificuldade com o arquivo para a arquitetura ARM, pois como é uma arquitetura diferente, tive de alterar algumas coisas para o formato da arquitetura, isto implicou em criar e recriar este arquivo de Dockerfile e do Docker-compose, algumas centenas de vezes, assim como a remoção das imagens e dos containers associados. Esta é uma prática comum durante a construção da imagem e dos containers, visto que a imagem é imutável e se for necessário fazer alterações nela, todo o processo necessitará ser reiniciado.

De acordo com Rafael Benevides – Diretor de Experiência de Desenvolvimento da Red Hat – Tradução nossa, “Containers são descartáveis”.

Cada comando executado dentro do Dockerfile, o Docker cria um sistema de camadas (layers) que o mesmo executa o comando e vai inserindo na imagem. O correto é que o Dockerfile seja o máximo otimizado, para que a imagem fique com poucas camadas de forma enxuta.

### **Principais instruções utilizadas no Dockerfile:**

- .dockerignore – Arquivo que contém informações que serão ignoradas pelo cliente do docker
- FROM – Informa a imagem base, subsequente para as instruções.
- MAINTAINER (depreciado – uso do LABEL) Informa o autor que gerou a imagem.

- RUN – Executa qualquer comando em uma nova camada da imagem e insere os resultados na imagem criada. Os resultados inseridos na imagem, serão usados no próximo passo do Dockerfile.
- CMD – Utilizado para prover argumentos para o comando ENTRYPOINT.
- EXPOSE – Expõe a lista de portas que poderão estar visíveis e acessadas externamente do host ao container.
- ENV – Informa variáveis de ambiente.
- ADD - Adiciona novos arquivos, diretórios ou remove arquivos do container. Este comando invalida o cache, para usar cache use o COPY.
- COPY – Copia novos arquivos, diretórios ao container. Este comando somente poderá ser executado como root, necessário fazer alteração de chown manualmente.
- ENTRYPOINT – Configura o container para executar comandos em shell (executáveis).
- VOLUME - Cria um mount point externo, que os containers irão guardar dados.
- WORKDIR – Seleciona o diretório padrão que será executados os próximos passos Dockerfile.
- LABEL – Aplicar chave/valor de metadados as suas imagens, containers ou deamons.

### **5.3 DOCKER IMAGEM**

A imagem do Docker é um pacote da aplicação, que acabou de ser criada (build) seguindo as configurações do arquivo Docker file. A imagem é criada antes de se criar os containers com os serviços; esses utilizam a imagem como base para prestarem seus serviços.

Mais de um container de serviço pode utilizar a mesma imagem, ou um único container de serviço pode ter a sua própria imagem.

De acordo com o livro Docker para desenvolvedores:

A imagem do Docker é uma abstração completa e não requer qualquer tratamento para lidar com as mais variadas distribuições GNU/Linux existentes, já que a imagem Docker carrega em si uma cópia completa dos arquivos de uma distribuição enxuta. [21]

Através do comando:

- docker images

Se torna possível visualizar no terminal as imagens baixadas no Docker local, conforme a tabela abaixo, do estudo de caso.

Tabela 1- Tabela de Imagens utilizadas no Estudo de Caso. Fonte: Próprio Autor.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
phalanx_app	Latest	68587ee939ad	4 days ago	2.12GB
phalanx_worker	Latest	68587ee939ad	4 days ago	2.12GB
Redis	3.2-alpine	0a216c0d97d9	5 days ago	19.8MB
Mysql	5.7	11615e225c92	6 days ago	408MB
portainer/portainer	Latest	771161a7316e	2 weeks ago	33MB
ledermann/base	Latest	c96b29ff3989	2 weeks ago	770MB
Ubuntu	12.04	5b117edd0b76	6 months ago	104MB
Ruby	2.3.3	0e1db669d557	7 months ago	734MB

A imagem é um container intermediário que faz a abstração das aplicações para o sistema operacional. A mesma é uma cama intermediária da qual é possível fazer instalação de bibliotecas que serão usadas como base para a criação dos containers das aplicações.

A imagem uma vez criada é imutável, a mesma não permite que sejam feitas alterações dentro dela. É possível entrar no container e fazer instalações de bibliotecas, dependências e etc; porém isto somente é válido enquanto o container estiver “up” se a instância do mesmo for destruída, tudo que fora instalado, executado dentro do mesmo se perde.

Há formas de se salvar dados, por volumes e logs pelas saídas específicas do drive do Docker, porém isto não se aplica à imagem.

Uma forma bem eficiente de construir a imagem é através da utilização do Docker-compose, pois isto criar várias imagens dos serviços escritos no arquivo de uma forma coordenada e reutilizar a mesma imagem para mais de um serviço. Esta é feita através da notação “&” na declaração do serviço dentro do arquivo.

A imagem, em minha opinião, é bastante útil e colaborativa, pois o Docker possui um repositório público/privado de imagens, chamado Docker Hub. Irei falar do mesmo mais adiante. Este repositório é possível fazer download de imagens e simplesmente utilizá-las.

O Download das imagens é feito pelo comando “docker pull”, comandos apresentados mais adiante.

Ao ser executado este comando, o Docker verifica se possui localmente a imagem, se não possuir o mesmo vai ao Docker Hub para poder fazer o download da imagem.

Primeiramente é necessário fazer o loggin no Docker Hub, pois o mesmo pode dar erro informando que não encontrou a imagem remotamente, sendo que falta o loggin do usuário.

Na construção da imagem é de suma interesse que a imagem fique de tamanho considerável, pois isto facilita se for feito push da mesma para o DockerHub.

### **Principais commandos da Imagem:**

- docker pull – Para fazer pulls de uma ou mais imagens do repositório para o cliente do docker na máquina local.
- docker push – Para fazer o push de uma ou mais imagens do Docker local para o repositório do docker Hub.
- docker images – Mostra as principais imagens que estão baixadas no Docker local.
- docker rmi <id\_imagem> – Comando para poder remover a imagem mencionada no ID.

## **5.4 DOCKERHUB**

O Docker disponibiliza um repositório público/privado para compartilhamento de imagens, que podem ser utilizados para viabilizar customizações de ambientes específicos e podem ser customizados pelos desenvolvedores, administradores de rede e etc.

As customizações podem ser feitas com alguns parâmetros passados ao instanciar o container e/ou utilizando arquivos de configurações específicos como o Docker-compose. Estas configurações, repassando parâmetros pode ser necessário para manter a adequação do ambiente à aplicação e/ou customização necessária.

As imagens no Docker Hub são identificadas através de tags, e assim permitir o armazenamento de múltiplas versões da mesma aplicação. As tags podem especificar a versão do build para aquela imagem; porém isto fica muito ao encargo de se ter seguido boas práticas de versionamento [22] para criar a tag da imagem:

- Major – Primeiro dígito
- Minor – Segundo dígito
- Build – Último dígito

Recomenda-se sempre usar imagens que são de repositórios oficiais, pois estas estão com as configurações padrão para o ambiente.

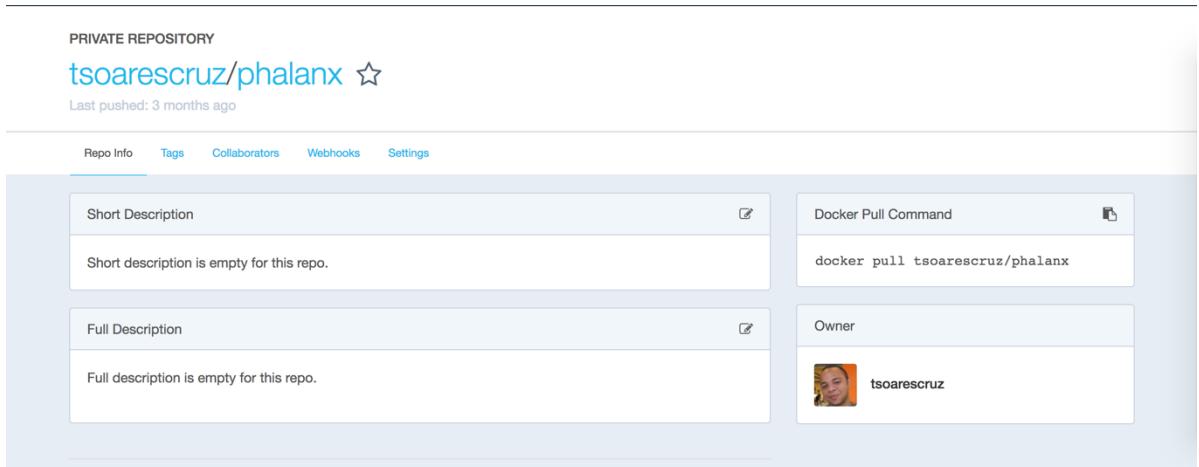


Figura 15 - Docker Hub do Projeto Fonte: <https://hub.docker.com/r/tsoarescruz/phalanx/>

### Explore Official Repositories

 nginx official	7.2K STARS	10M+ PULLS	<a href="#">DETAILS</a>
 alpine official	2.7K STARS	10M+ PULLS	<a href="#">DETAILS</a>
 redis official	4.4K STARS	10M+ PULLS	<a href="#">DETAILS</a>
 busybox official	1.1K STARS	10M+ PULLS	<a href="#">DETAILS</a>
 httpd official	1.3K STARS	10M+ PULLS	<a href="#">DETAILS</a>
 ubuntu official	6.8K STARS	10M+ PULLS	<a href="#">DETAILS</a>

Figura 16 – Explorer do Docker Hub Fonte: <https://hub.docker.com/explore/>

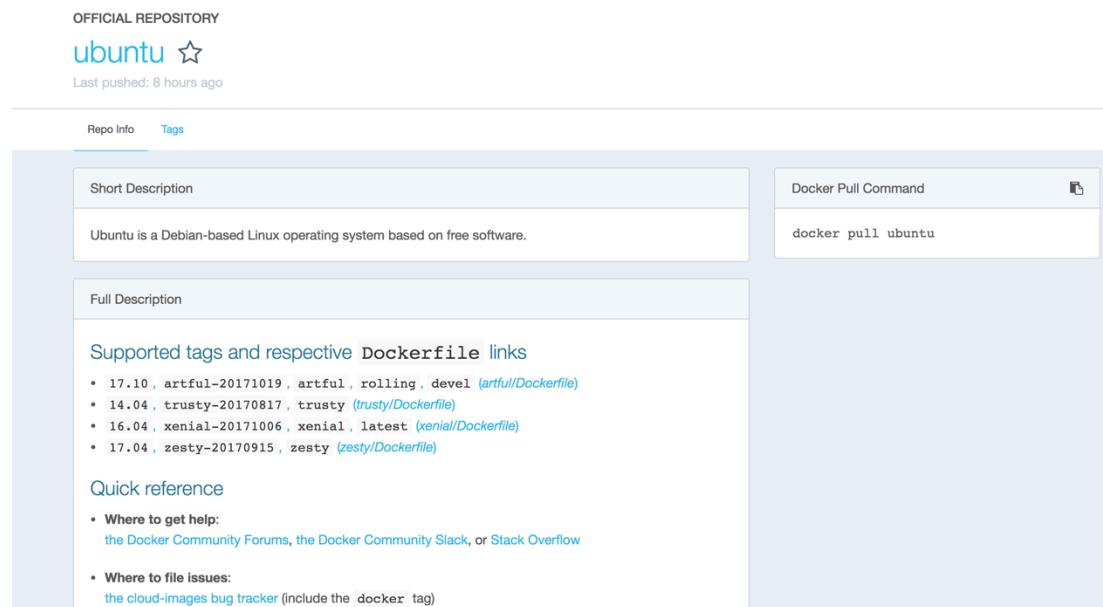


Figura 17 – Repositório oficial da imagem do Ubuntu Fonte: [https://hub.docker.com/\\_/ubuntu/](https://hub.docker.com/_/ubuntu/)

A possibilidade de baixar software (imagem) é um dos conceitos que viabiliza muito o conceito de virtualização e de software como serviço; pois não é mais necessário ficar fazendo instalações de softwares na infraestrutura, o mesmo é mais um serviço rodando em conjunto com a aplicação.

Através do comando:

- Docker pull <repositório da imagem>

É possível baixar a imagem para o cliente do docker local e poder construir “build” os containers de serviço com esta imagem associada.

### **Principais comando do Repositório:**

- docker login – Para fazer loggin no repositório.
- docker logout – Para fazer logout do repositório.
- docker search – Para fazer pesquisas nos registros das imagens no repositório.

## **5.5 DOCKER CONTAINER**

Uma das definições que encontrei na documentação oficial [23]:

Tradução nossa, “Container é uma instância em tempo de execução da imagem.”

O Docker container consiste:

- Docker imagem
- Ambiente de execução
- Instruções principais

Conforme em capítulos anteriores desta publicação o Docker container utiliza todos os conceitos do LXC container. Sendo o mesmo uma conjunto de instruções formado pela construção da imagem, ou download da imagem do Docker Hub e tem como premissa a execução de um serviço.

A tabela a seguir mostra todos os containers de serviço existentes no meu estudo de caso. Todos os parâmetros para a construção do container são definidos nos arquivos do Docker file e Docker-compose o mesmo somente é uma instrânciâcia de execução do serviço descreitos nesses arquivos.

A tabela foi informada através da execução do comando:

- docker container ls

Tabela 2 - Tabela de containers utilizadas no Estudo de Caso. Fonte: Próprio Autor

CONTAINER ID	IMAGE	COMMAND	CREATE	STATUS	PORTS	NAMES
c7cc8808836d	phalanx_worker	"bundle exec sidekiq"	4 days ago	Up 4 days	80/tcp, 300/tcp, 9000/tcp	phalanx_worker_1
caf0ecd83c4f	phalanx_app	"bundle exec rails..."	4 days ago	Up 4 days	80/tcp, 300/tcp, 9000/tcp, 0.0.0.0:300	phalanx_app_1
3c45af618d36	redis:3.2-alpine	"docker-entrypoint..."	4 days ago	Up 4 days	6379/tcp	phalanx_redis_1
ccad4b9d7b7b	portainer/portainer	"/portainer"	4 days ago	Up 4 days	0.0.0.0:300 2->9000/tcp	phalanx_ui_1
bbcae433ef46	mysql:5.7	"docker-entrypoint..."	4 days ago	Up 4 days	0.0.0.0:330 7->3306/tcp	phalanx_db_1

Os parâmetros mais utilizados na execução do container são:

Tabela 3 - Tabela de parâmetros utilizado na manipulação dos containers Fonte:  
<https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/comandos.md>

Parâmetro	Explicação
-a	Lista todos os containers, inclusive os desligados
-l	Lista os últimos containers, inclusive os desligados
-n	Lista os últimos N containers, inclusive os desligados
-q	Lista apenas os ids dos containers, ótimo para utilização em scripts

Containers são muito instáveis e descartáveis, durante o meu estudo de caso, tive que descartar várias vezes alguns containers de serviço, pois os mesmos são dependentes entre si e se um serviço não for instânciado de forma correta o container dependente também será afetado. Já fiz a descrição desta dependência durante esta obra.

Isto entra em total acordo com a informação do Rafael Benevides – Diretor de Experiência de Desenvolvimento da Red Hat [13] – Tradução nossa, “Containers são descartáveis”.

### **Principais comandos de manipulação dos container:**

- docker create – cria um container, porém não inicia o mesmo.
- docker rename – Renomeia o label do container.
- docker run – Cria o container e inicia o mesmo, coloca em operação.
- docker rm <nome do container> - Deleta o container informado pelo nome.
- docker update – Faz um update das configurações do container.
- docker inspect <id\_container> - Informa dados detalhados do container.
- docker container prune – Remove todos os containers que não estão em execução no momento.

Todos esses comandos [24] permitem parâmetros de configurações, como os descritos no arquivo do Docker-compose; pois é possível fazer a execução de um container sem a utilização de um arquivo do Docker-compose ou Docker file, porém se executado desta forma todos os argumentos devem ser passados de forma explícita na criação ou execução do container, da seguinte sintaxe:

- docker container run <parâmetros> <imagem> <CMD> <argumentos>  
Não sendo uma boa prática executar desta forma, conforme o exemplo:
- docker run -it -d -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock alexellis2/visualizer-arm

Os parâmetros mais utilizados na execução do container são:

Tabela 4 - Tabela de parâmetros utilizados na execução dos containers Fonte:

<https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/comandos.md>

Parâmetro	Explicação
-d	Execução do container em background
-i	Modo interativo. Mantém o STDIN aberto mesmo sem console anexado
-t	Aloca uma pseudo TTY
--rm	Automaticamente remove o container após finalização (Não funciona com -d)
--name	Nomear o container
-v	Mapeamento de volume
-p	Mapeamento de porta
-m	Limitar o uso de memória RAM
-c	Balancear o uso de CPU

Segue um exemplo simples no seguinte comando:

- docker container run -it --rm --name phalanx\_app\_1 ruby bash

### 5.5.1 Software de Gerenciamento de Containers

Em meu estudo de caso, fiz uso de um serviço open source Portainer [25]. Esse serviço de UI me permite fazer todos os gerenciamento com extrema facilidade:

- Gerenciamento dos Serviços
- Gerenciamento dos Containers
- Gerenciamento de Logs
- Gerenciamento das Imagens
- Gerenciamento de Rede
- Gerenciamento de Volumes
- Gerenciamento do Swarm

Esta UI, me permite fazer todo o gerenciamento e execuções dos containers de forma prática, porém todo esse gerenciamento pode ser executado através de linhas de comando. Essa UI acessa a API do Docker para fazer esse gerenciamento.

Segue as imagens do serviço:

The screenshot shows the Portainer.io dashboard with the following information:

- Node info:**
  - Name: moby
  - Docker version: 17.09.0-ce
  - CPU: 2
  - Memory: 3.1 GB
- Containers:** 5 total, 4 running, 0 stopped.
- Images:** 7 total, 4.2 GB.
- Volumes:** 4 total.
- Networks:** 4 total.

Figura 18 - Dashboard do Portainer Fonte: Próprio autor

The screenshot shows the Portainer.io Images dashboard with the following interface and data:

- Pull image:** Search bar for "Name" (e.g., myImage:myTag) and Registry (DockerHub).
- Images:** A table listing images:
 

ID	Tags	Size	Created
sha256:5b117edd0b...	ubuntu:12.04	103.6 MB	2017-04-12 18:05:30
sha256:0e1db69d5...	ruby:2.3.3	733.9 MB	2017-03-22 21:26:25
sha256:0a216c0d97...	redis:3.2-alpine	19.8 MB	2017-10-26 01:06:34
sha256:771161a731...	portainer/portainer:latest	33 MB	2017-10-15 16:43:40
sha256:e60194d02e...	phalconx/app-test	2.1 GB	2017-10-26 18:39:48
sha256:11619e225c...	mysql:5.7	408.2 MB	2017-10-25 17:11:08
sha256:c96d929ffcf39...	ledermann/base:latest	770.2 MB	2017-10-11 07:02:20

Figura 19 - Dashboard de Imagens do Portainer Fonte: Próprio autor

Figura 20 - Dashboard de Volumes do Portainer Fonte: Próprio autor

Figura 21 - Dashboard de Engine do Portainer Fonte: Próprio autor

Essa UI é interessante para usuários leigos, pois a sua usabilidade é ótima e permite fazer interações com as imagens containers em poucos passos.

O docker possui uma versão empresarial Docker EE que possui uma interface de gerenciamento própria. Não obtive informações de custo dessa versão empresarial, pois o docker não libera valores e sim um canal de contato com consultores especializados, toda a comunicação é em inglês; e os mesmos prestam um serviço todo especializado para empresas.

## 5.6 DOCKER SWARM

O docker Swarm é a implementação do Docker para cluster.

De acordo com a documentação oficial do Docker, o Docker Swarm [26] seria:

Tradução nossa, “Docker Swarm é o nome da ferramenta de clusterização nativa do Docker. O Docker Swarm agrupa vários hosts do Docker e os expõe como um único host virtual do Docker. Ele serve a API padrão do Docker, então qualquer ferramenta que já funciona no Docker pode agora se escalar de forma transparente para múltiplos hosts”.

Nesta implementação é elegível um host (node) master e a partir dele são elegíveis os nós escravos. Ambos os nós tem que estar na mesma rede física e virtual (podendo ser criada dentro do Docker).

Todos os comandos do cluster são sempre executados dentro do nó master. É possível eleger um outro nó se o master cair (ficar down). Este nó fica no estado elegível.

Para inserir um host no swarm é necessário executar o comando:

- docker swarm init

A saída deste comando irá ter um token de acesso é a interface de rede que será visível para ser agrupada para os outros hosts:

- docker swarm join --token SWMTKN-1-05fjhstn68ea0d6njhx8zgqrsxq47jznacn2niku4zqq2vsrqq-5ytikrhfg1tujjxpju0o6q8qe 192.168.65.2:2377

Ao executar este comando nos outros hosts que estão com o docker instalado e que podem ser visíveis para o host que executou o comando (rede física e virtual), os mesmos se tornam nodes escravos do node máster.

Como já dito anteriormente os mesmos não executaram mais comandos da aplicação e somente executarão os serviços que lhes forem informados para a execução conforme o arquivo do docker-compose versão 3:

- Docker-compose versao 3 : Apêndice 13.5

Em meu estudo de caso eu fiz a implementação deste arquivo direto no cluster de serviço próprio através do comando:

- docker stack deploy --compose-file docker-compose\_3\_version.yml phalanx

Este comando irá fazer o deploy dos serviços, conforme descrito no docker-compose. Diferente da versão 2, neste formato é possível fazer para cada serviço, associando informações de reserva de CPU, memória e alocação do nó que irá rodar o serviço, conforme trecho abaixo:

```
#service deployment
deploy:
  mode: replicated
  replicas: 1
  labels: [APP=PHALANX]
#service resource management
resources:
  #Hard limit - Docker does not allow to allocate more
  limits:
    cpus: '2.25'
    memory: 512M
  #Soft limit - Docker makes best effort to return to it
  reservations:
    cpus: '1.25'
    memory: 256M
#service restart policy
restart_policy:
  condition: on-failure
  #delay: 5s
  max_attempts: 10
  window: 120s
#service update configuration
update_config:
  parallelism: 1
  delay: 5s
  failure_action: continue
```

```

monitor: 10s
max_failure_ratio: 0.3
#placement constraint - in this case on 'worker' nodes only
placement:
constraints: [node.role == manager]

```

Para verificação dos serviços disponíveis em cada nó, fiz a utilização de um serviço open-source Visualizer-arm [27], o mesmo faz o monitoramento de todos os nós, que estão na mesma rede (física e virtual) e informa todos os containers que estão rodando em cada nó.

Segue a imagem de utilização do serviço:

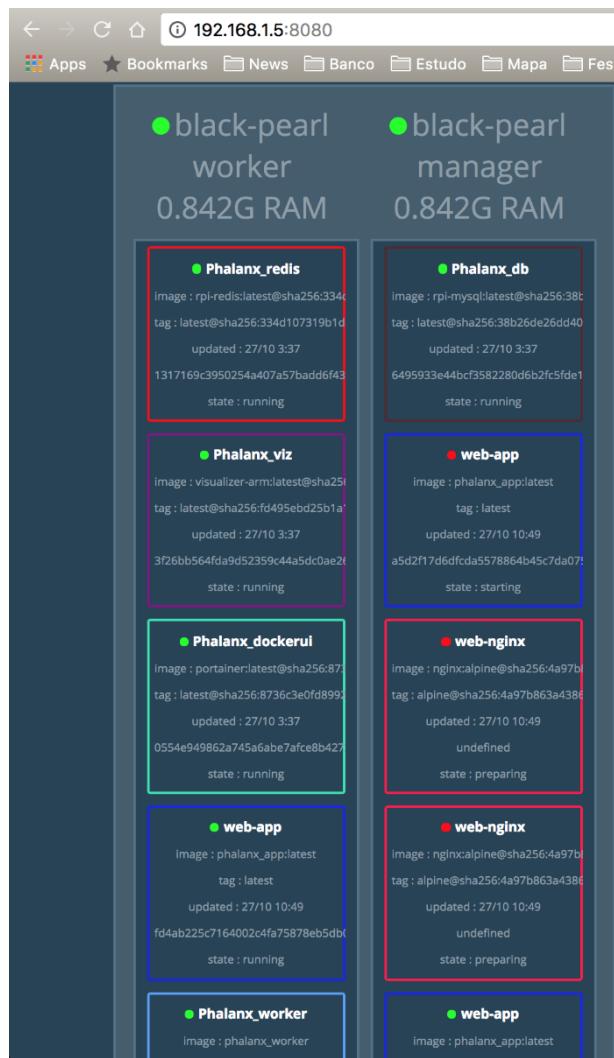


Figura 22 - Visualizer-arm Fonte: Próprio autor

Na implementação do Swarm também é possível fazer a utilização de arquivo Docker-compose ou fazer a instância do serviço sem os arquivos de configuração. Sendo necessário passar todos os comandos de forma explícita, conforme o exemplo:

- docker service create --name web-nginx --replicas 4 --restart-max-attempts 3 - -restart-window 5s --rollback-delay 3s --workdir /myapp/ -p 8080:80 nginx:alpine

## 5.7 PLAY WITH DOCKER

O Docker possui uma engine de testes do seu próprio laboratório.<sup>1</sup>

Essa engine consiste em uma máquina virtual com Alpine Linux; foi criada pelos capitães do Docker Marcos Nils e Jonathan Leibiusky, sendo possível fazer testes, alterações, criação e configurações em containers do docker, imagens e do módulo Swam do docker em alguns minutos. [28]

Maiores informações podem ser obtidas no github do projeto [29]; Este projeto também é utilizado para treinamentos do Docker.

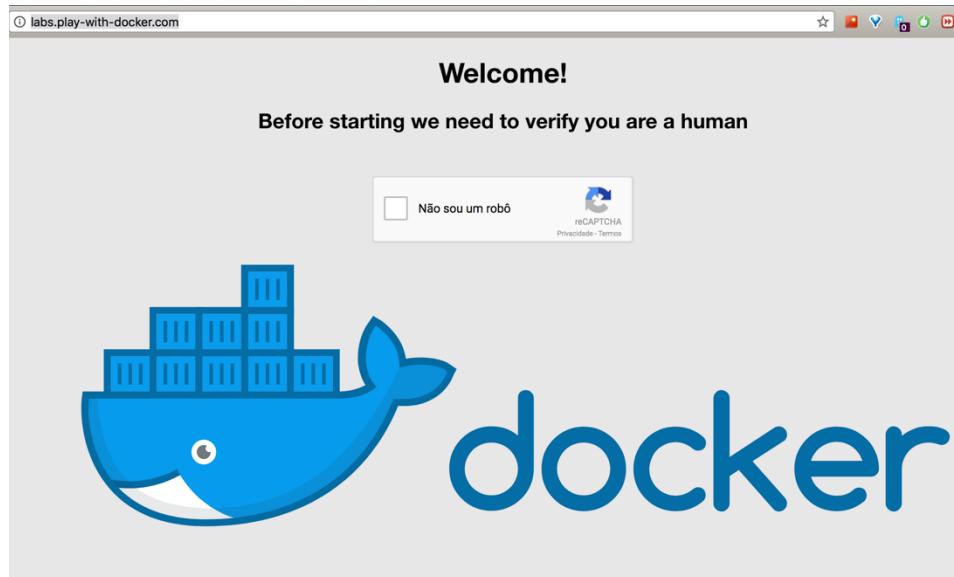


Figura 23 - Play with Docker Fonte: <http://labs.play-with-docker.com/>

O projeto conta com uma documentação[30] simples que ajuda bastante. Existem outros treinamentos do docker.

Este projeto também é bastante útil, para testes sem a necessidade de execução do docker em próprio host, é possível utilizar instâncias remotas dentro do projeto. Para tal é necessário a instalação de um drive:

---

<sup>1</sup> <http://labs.play-with-docker.com>.

- “docker-machine-drive-pwd”

Este drive está disponível no github:

- <https://github.com/play-with-docker/docker-machine-driver-pwd/releases>

Através do comando: docker-machine create -d pwd --pwd-url <url-do-play-with-docker>

Exemplo de utilização:

- docker-machine create -d pwd --pwd-url [https://labs.play-with-docker.com/p/b7uf58pe0c5g00d1jutg#b7uf58pe\\_b7uf5c1e0c5g00d1juu0](https://labs.play-with-docker.com/p/b7uf58pe0c5g00d1jutg#b7uf58pe_b7uf5c1e0c5g00d1juu0) node1

Este projeto possui uma interface bastante simples possibilitando que possam ser criadas novas instâncias de forma gráfica e amigável.

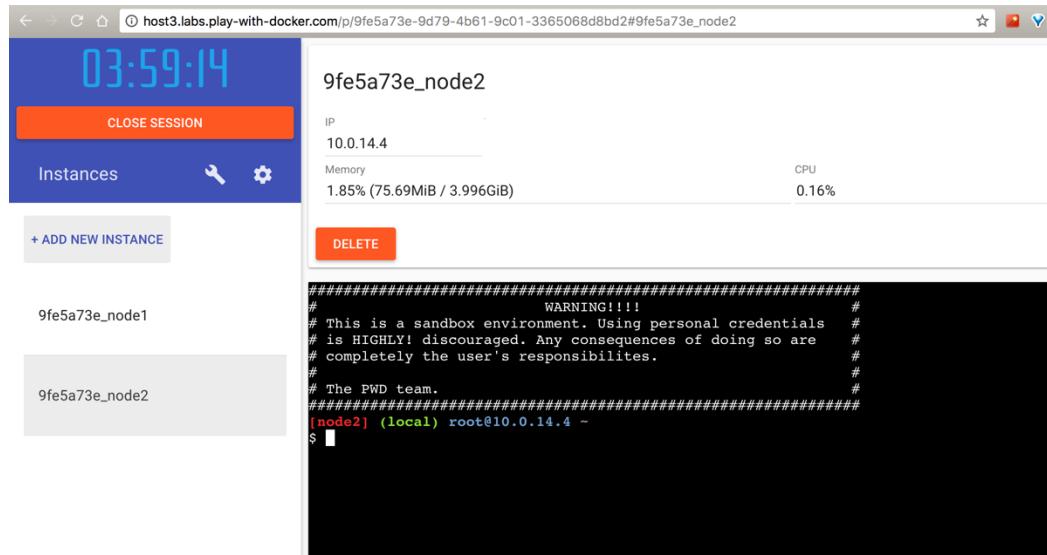


Figura 24 - Shell do Play with Docker Fonte: [http://host3.labs.play-with-docker.com/p/9fe5a73e-9d79-4b61-9c01-3365068d8bd2#9fe5a73e\\_node2](http://host3.labs.play-with-docker.com/p/9fe5a73e-9d79-4b61-9c01-3365068d8bd2#9fe5a73e_node2)

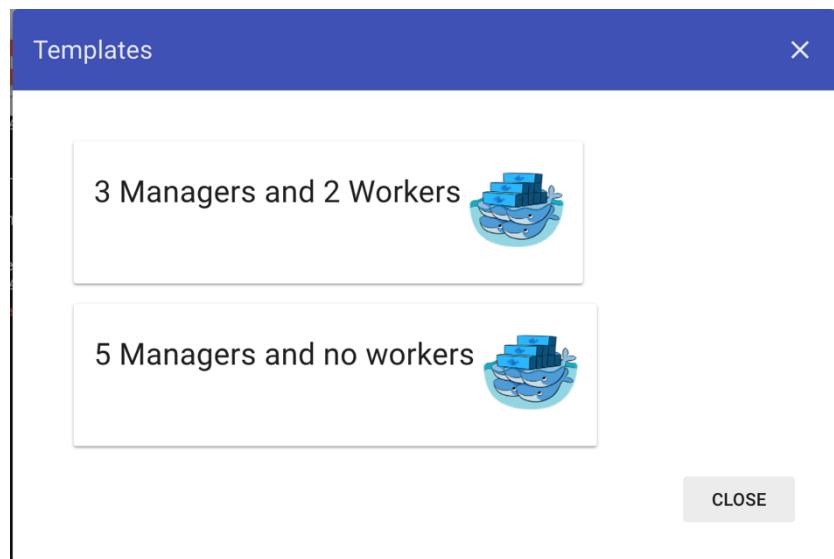


Figura 25 - Template do Play with Docker Fonte: [http://host3.labs.play-with-docker.com/p/9fe5a73e-9d79-4b61-9c01-3365068d8bd2#9fe5a73e\\_node2](http://host3.labs.play-with-docker.com/p/9fe5a73e-9d79-4b61-9c01-3365068d8bd2#9fe5a73e_node2)

## 5.8 COMUNIDADE E EMPRESARIAL

Como o Docker é uma plataforma open source, ele já possui esta característica de comunidade e contribuições.

Durante o meu estudo de caso eu participei da comunidade do Docker no telegram e no Slack e percebi que esta comunidade, em ambas as ferramentas de comunicação, foi uma das mais ativas que já obtive contato.

Pessoas que já utilizam o Docker em seus ambientes, sejam de teste ou em produção sempre tiram dúvidas das pessoas mais iniciantes e leigas no assunto. Recomendo fortemente a participação desta comunidade em ambas as ferramentas de comunicação.

A comunidade é bem grande e chega até a impressionar pelos números de contribuidores.

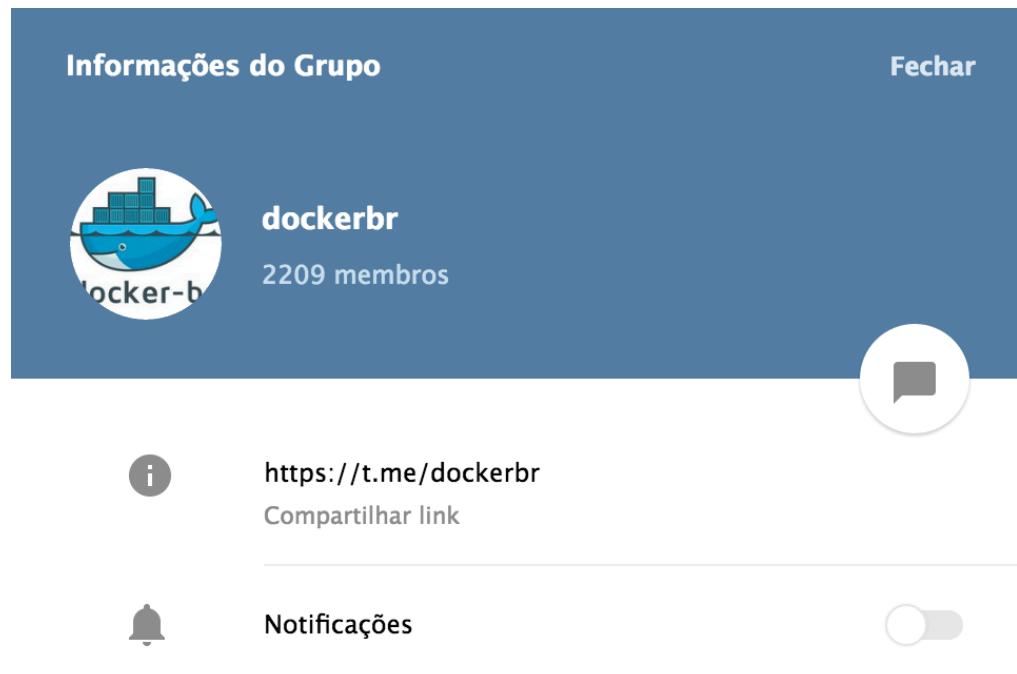


Figura 26 - Comunidade do Docker no Telegram Fonte: <https://t.me/dockerbr>

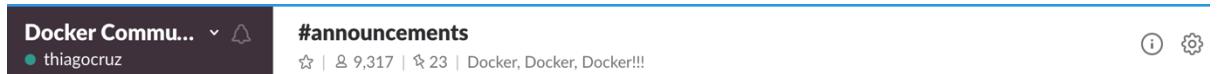


Figura 27 - Canal Annoucements do Slack Fonte: Próprio autor

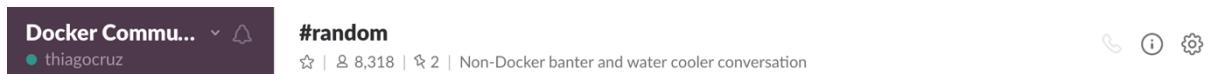


Figura 28 - Canal Random do Slack Fonte: Próprio autor

O Docker possui anualmente uma conferência a *Docker Conf*[32] , da qual são informados novas releases e informações sobre a plataforma, todas as informações são em Inglês. Fortalecendo mais ainda o pensamento de comunidade e compartilhamento de informações.

Esse compartilhamento já é nativo do conceito do Docker, visto que possui repositório público e privado de imagens (*DockerHub*), já citado na obra, com o conceito de compartilhamento de imagens já criadas de forma a fomentar esse conceito.

### 5.8.1 Empresarial

O Docker possui uma distribuição própria para empresas o Docker EE, já fiz comentários sobre o mesmo durante a execução desta obra.

Esta versão da plataforma é de utilização exclusiva para *Data Centers*, e para clientes corporativos, contando com um dashboard de supervisão de containers, sendo capaz de provê serviço de multi-orquestração e escalonamento dos containers (serviços) do cliente além de serviços exclusivos de segurança e monitoramento do ambiente do cliente, tendo todo o ciclo de vida (*deploy*) de gerenciamento com suporte.

Esta versão conta também com uma loja [33], da qual é possível comprar imagens certificadas e alguns templates prontos para deploy como o EC2 AWS [34].

Em meu estudo sobre esta obra, não obtive a oportunidade de fazer verificações sobre esta versão do Docker EE, apesar de ter um período de testes (trial), não fiz verificações sobre esta versão, porém a mesma é utilizado por alguns consumidores famosos do Docker, como algumas universidades Norte-Americanas e algumas indústrias.

Em um congresso de tecnologia obtive informações que algumas empresas de IT bem consolidadas no mercado, que utilizam o Docker em suas plataformas de Cloud, como a IBM® [35]; A plataforma Bluemix da IBM®[36] (*PaaS*) utiliza o Kubernetes que é baseado na tecnologia de Docker container. A mesma até permite fazer deploy da imagem gerada no Docker diretamente para a plataforma do Bluemix®.

## **6 BOAS PRÁTICAS DE CONSTRUÇÃO DA APLICAÇÃO (DOZE FATORES)**

Esses doze fatores, são fatores para a melhor criação, manutenção, atualização e entregas de softwares como serviços que são denominados *web apps*, ou *software como serviço*.

Conforme Rafael Gomes (docker para desenvolvedores) uma vez que sua aplicação siga todas as boas práticas apresentadas neste documento, você possivelmente estará usando todo potencial que o Docker tem a lhe proporcionar.[15]

Conforme informações do oficiais, segue uma breve descrição dos doze (12) fatores:

A aplicação doze fatores é uma metodologia para construir softwares como serviço que preza utilizar:

- Formatos declarativos para automatizar a configuração inicial, minimizar tempo e custo para novos desenvolvedores participarem do projeto;
- Tem um contrato claro com o sistema operacional que o suporta, oferecendo portabilidade máxima entre ambientes que o executem;
- São adequados para implantação em modernas plataformas em nuvem, evitando a necessidade por servidores e administração do sistema;
- Minimizam a divergência entre desenvolvimento e produção, permitindo a implantação contínua para máxima agilidade;
- E podem escalar sem significativas mudanças em ferramentas, arquiteturas, ou práticas de desenvolvimento.

A metodologia “doze fatores” pode ser aplicada a aplicações escritas em qualquer linguagem de programação, e que utilizem qualquer combinação de serviços de suportes: banco de dados, filas, cache de memória e etc.

### **6.1 OS DOZE FATORES**

- I. Base de Código - Uma base de código com rastreamento utilizando controle de revisão, muitos deploys
- II. Dependências - Declare e isole as dependências
- III. Configurações - Armazene as configurações no ambiente

- IV. Serviços de Apoio - Trate os serviços de apoio, como recursos ligados
- V. Build, release, run - Separe estritamente os builds e execute em estágios
- VI. Processos - Execute a aplicação como um ou mais processos que não armazenam estado
- VII. Vínculo de porta - Exporte serviços por ligação de porta
- VIII. Concorrência - Dimensione por um modelo de processo
- IX. Descartabilidade - Maximizar a robustez com inicialização e desligamento rápido.
- X. Ambientes (*DEV/PROD*) semelhantes - Mantenha o desenvolvimento, teste, produção o mais semelhante possível.
- XI. Logs - Trate logs como fluxo de eventos.
- XII. Processos de Admin - Executar tarefas de administração/gerenciamento como processos pontuais.

## 7 SOFTWARES DE ORQUESTRAÇÃO

Existem algumas Plataformas (*PaaS*) para gerenciamento de containers e serviços nos ambientes (*QA*, *STAGING* e *PROD*).

Essas plataformas tem a finalidade de melhorar o condicionamento de containers, de forma visual e prática, para que seja abstraída a complexidade de gerenciamento e que não seja necessários conhecimentos de comandos, e entre outras finalidades.

Dentre estas plataformas se destacam algumas, como: Tsuru, Kubernet é Vagran.

Essas plataformas além de melhorarem o gerenciamento, provisionamento e condicionamento dos containers elas melhoraram o acesso ao conhecimento e a popularização do tema; pois possibilitam que pessoas com poucos conhecimentos, profundos sobre certas áreas, possam verificar/monitorar um ambiente com serviços por container.

Um outro provedor/empresa que alavancou esse conhecimento sobre virtualização, foi a Amazon®, com seus serviços em AWS®; os mesmos são segregados em agregação de serviços e monetização por usabilidade dos mesmos.

Algumas dessas *PaaS*, citadas possuem integração com a AWS®.

Em meu estudo de caso não obtive muito contato com essas *PaaS*, todo o meu trabalho foi em cima da ferramenta nativa do Docker.

Porém fiz uma entrevista com o Coordenador de Produtos de Cloud da empresa de internet de um grande grupo de mídias brasileiro, da qual trabalho. Não obtive autorização para revelar o nome da empresa.

O mesmo utiliza a *PaaS* do Tsuru, em infra própria; porém fazem deploy em cloud com o Keubernets® e na AWS; fazem deploys de testes para assegurar que o serviço está disponível de redes externas e apontam para a própria infraestrutura.

O mesmo me informou que utilizam arquivos de configuração no Kubernets® com a mesma finalidade do Docker-compose e com integração com CI, fazendo orquestração dos serviços e utilizando o conceito de Blue Green.

Este conceito de Blue Green é muito usual para deploys em containers, pois há a possibilidade de ter duas versões da aplicação em produção ao mesmo tempo, ou como me foi explicado nesta entrevista, no arquivo do Kubernets® há a possibilidade de ser configurado um tempo de permanência (tempo de vida) do container antigo da aplicação, o mesmo pára de receber requisições e o outro container com o deploy atual da aplicação é iniciado, podendo ou não haver down time da aplicação.

## **8 ESTUDO DE CASO**

### **8.1 OBJETIVO**

Para este estudo de caso, fiz utilização do aprendizado da cadeira Engenharia de Software, o propósito deste documento é coletar, analisar e definir as necessidades de alto-nível e características do sistema, focando nas potencialidades requeridas para o correto funcionamento e informativo aos usuários-alvo.

Visa documentar o sistema e o ambiente geral, fornecendo a todos os envolvidos uma descrição comprehensível deste e suas macro funcionalidades.

### **8.2 CENÁRIO ATUAL**

Para a demonstração da aplicabilidade da tecnologia de container utilizando a plataforma Open-Source do Docker,

Foi desenvolvido e alterado este crawler de internet, para que o mesmo possa utilizar a tecnologia de container com serviços associados.

Cada container é um serviço diferente e específico que provê serviços a outros serviços.

Este sistema utiliza do conceito de produtos chaves e tags, palavras associadas para que possam ser buscadas no Google, o produto e a tags associada ao mesmo.

### **8.3 DESCRIÇÃO DO PROJETO**

O sistema irá permitir que qualquer usuário válido, cadastre uma tag e um produto. Os produtos serão pesquisados no Google repassando as tags informadas. O Sistema dará a possibilidade de fazer a associação de tags e produtos. Existe a possibilidade de ver dados da consulta na tela de consultas.

Este sistema de buscas (crawler) tem como finalidade fazer buscas no Google, sobre determinados produtos (palavras), e suas tags (atributos) associados.

O mesmo foi criado sobre o paradigma de programação por serviço, utilizando containers do Docker, sendo quarto (4) containers de serviço:

- Data base: Serviço de Banco de Dados em MySQL. Sendo o Banco de Dados permanente para a aplicação

- Redis: Serviço de Banco de Dados no Redis-Server. Sendo o Banco de Dados volátil da aplicação, utilizado para armazenar as buscas.
- Sidekiq: Serviço enfileiramento de tarefas, da qual é utilizado para poder fazer as buscas na web.
- Phalanx: Aplicação em Ruby, da qual é responsável por fazer as buscas na web

Há a possibilidade de excluir tags e produtos já cadastrados, pois ambos ficam em histórico em banco de dados.

O Sistema poderá exportar dados técnicos das consultas para três formatos, escolhidos pelo usuário.

O Sistema já possui uma base de dados inicial cadastrada, com algumas tags e produtos iniciais, para a sua execução.

## 8.4 ENVOLVIMENTO

### 8.4.1. Abrangência

O sistema é desenvolvido para poder rodar em duas arquiteturas distintas X64 e ARM, sendo um sistema Web pensado para poder fazer pesquisas na web e tags e produtos cadastrados.

Possui abrangência para o cluster de Raspberry Pi 3 que está rodando e qualquer computador de arquitetura X64, que fizer download do projeto.

## 8.5 RESTRIÇÕES

- **Funcionais/Negócio** – O sistema deverá permitir a visualização de todas as buscas que já foram realizadas, podendo fazer filtros nas buscas.
- O sistema deverá permitir que outros usuários consigam ver as buscas geradas por outro usuário.
- O sistema não executará a consulta se não houver nenhuma tag associada ao produto.
- O sistema deverá permitir que seja possível a exportação das gerado várias versões de provas, de acordo com o escolhido pelo usuário.
- **Tecnológicas** – O sistema só poderá ser executado utilizando a plataforma do Docker.
- **Operacionais** – Haverá somente um usuário, administrador para efetuar possíveis ações no sistema.

## **8.6 PROPOSTA DE SOLUÇÃO TECNOLÓGICA ESCOLHIDA**

Foi escolhido uma aplicação em Ruby, com framework Rails. Para realizar as tarefas de busca, será utilizado o schedule Sidekiq, e o banco de dados de alto desempenho em memória Redis. Será utilizado o banco de dados MySQL, para guardar os dados.

## **8.8 DIAGRAMA DE ATIVIDADES**

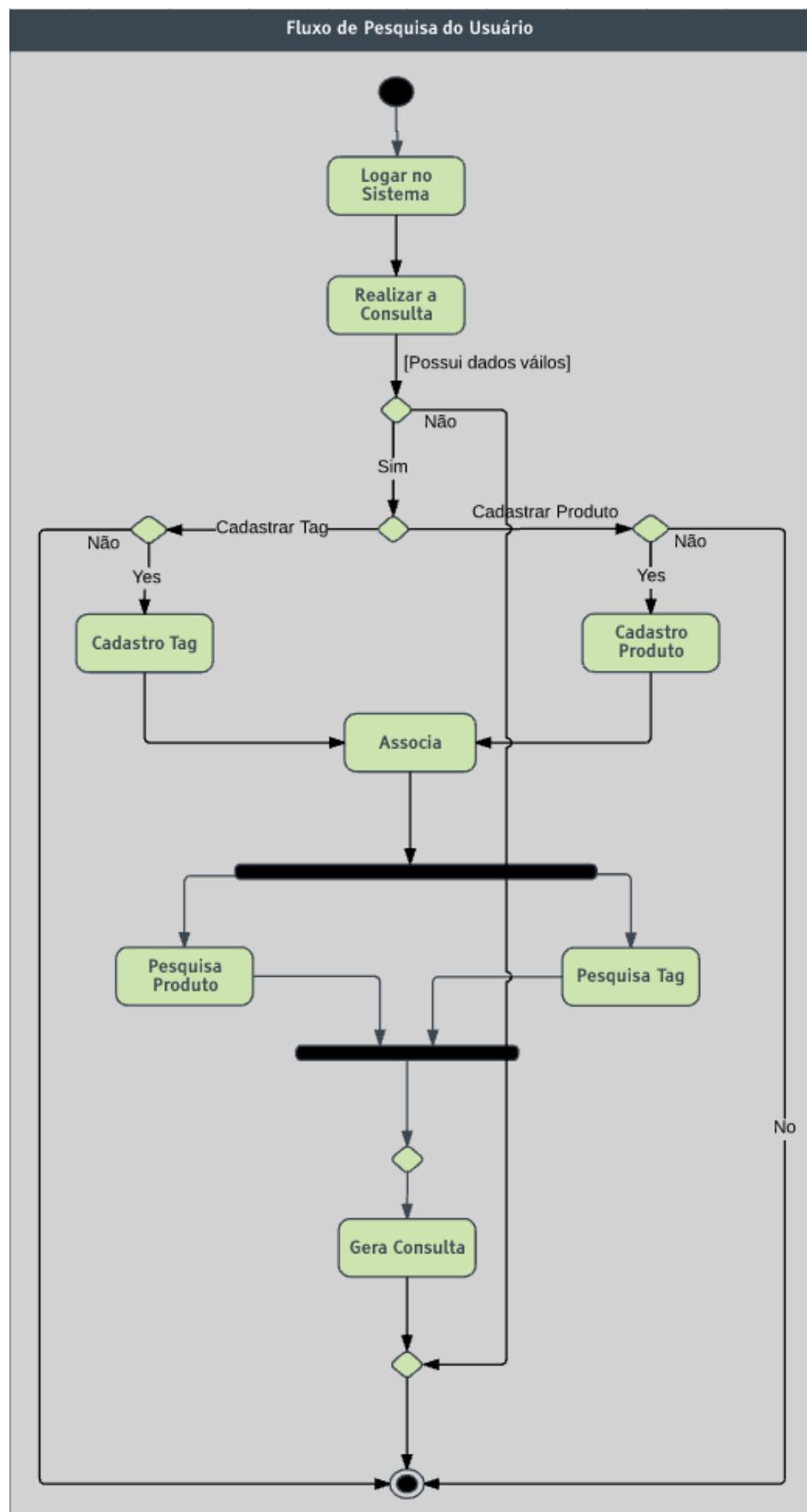


Figura 29 - Diagrama de Atividades do Estudo de Caso  
 Fonte:  
<https://www.lucidchart.com/documents/edit/714340a7-522c-4681-9605-41ae5d58fc02#>

## **8.9 REGRAS DE NEGÓCIO**

### **RN1 Acesso ao Sistema**

**RN1.1** Somente o Administrador poderá cadastrar usuário no sistema:

- Todos os usuários deverão ser cadastrados pelo primeiro usuário já cadastrado no Banco de Dados, tendo o perfil e função de Administrador.

**RN1.2** Usuário só entra no sistema com e-mail e senha:

- Os usuários são identificados por e-mail e senha para acesso ao sistema. Dentro do sistema, não há separação de perfil de usuários;
- Cabe ao primeiro usuário cadastrado, tendo a responsabilidade de fazer o cadastro dos outros usuários;
- O cadastro de usuários somente pode ser feito, já logado no sistema e utilizando o próprio sistema.

### **RN2 Ações no sistema**

**RN2.1** Perfil de Acesso:

- Não há separação de perfil de acesso no sistema.

### **RN3 Ações Gerais no sistema**

**RN3.1** Consultar Produtos:

- Todos os usuários, ativos e logados ao sistema poderão fazer consulta de produtos.

**RN3.2** Consultar Tags:

- Todos os usuários, ativos e logados ao sistema poderão fazer consulta de tags.

**RN3.3 Fazer Consultas:**

- Todos os usuários, ativos e logados ao sistema poderão fazer consultas no sistema.

**RN3.4 Adicionar Produtos:**

- Todos os usuários, ativos e logados ao sistema poderão adicionar produtos.

**RN3.5 Adicionar Tags:**

- Todos os usuários, ativos e logados ao sistema poderão adicionar tags.

**RN3.6 Remover Produtos:**

- Todos os usuários, ativos e logados ao sistema poderão remover produtos no sistema.

**RN3.7 Remover Tags:**

- Todos os usuários, ativos e logados ao sistema poderão remover tags no sistema.

**RN3.8 Gerar Relatório:**

- Todos os usuários, ativos e logados ao sistema poderão gerar relatórios;

**RN3.9 Exportar Relatório:**

- Todos os usuários, ativos e logados ao sistema poderão exportar relatórios gerados;

## **8.10 INTERFACE VISUAL**

### **8.10.1. Alta Fidelidade**

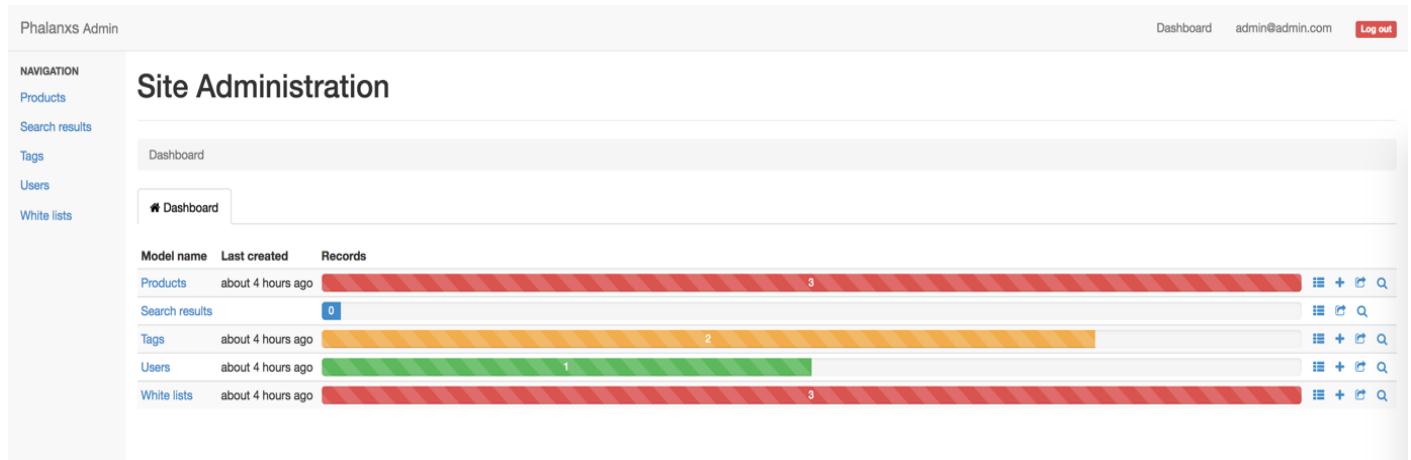


Figura 30 - Dashboard Principal do Estudo de Caso Fonte: Próprio autor

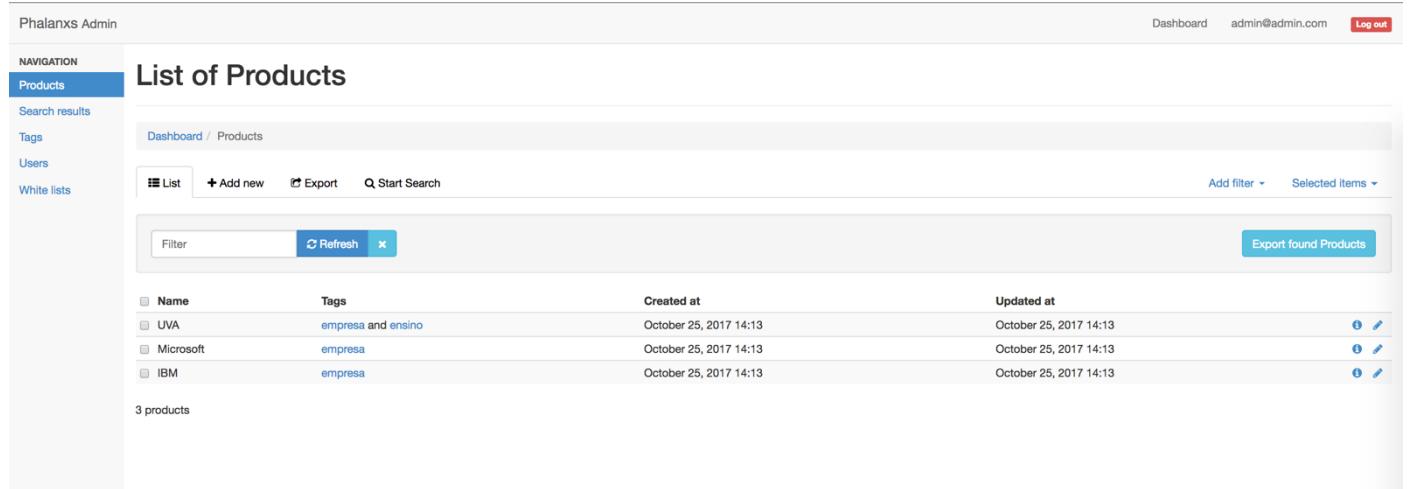


Figura 31 - Dashboard de Produtos do Estudo de Caso Fonte: Próprio autor

Phalanxs Admin

Dashboard admin@admin.com Log out

**List of Tags**

NAVIGATION  
Products  
Search results  
**Tags**  
Users  
White lists

Dashboard / Tags

List Add new Export Start Search Add filter Selected items

Filter Refresh Export found Tags

Id	Name	Created at	Updated at	Products
2	ensino	October 25, 2017 14:13	October 25, 2017 14:13	UVA
1	empresa	October 25, 2017 14:13	October 25, 2017 14:13	IBM, Microsoft, and UVA

2 tags

Figura 32 - Dashboard de Tags do Estudo de Caso Fonte: Próprio autor

Phalanxs Admin

Dashboard admin@admin.com Log out

**List of White lists**

NAVIGATION  
Products  
Search results  
Tags  
Users  
**White lists**

Dashboard / White lists

List Add new Export Start Search Add filter Selected items

Filter Refresh Export found White lists

Id	Domain	Created at	Updated at
3	uva.br	October 25, 2017 14:13	October 25, 2017 14:13
2	microsoft.com	October 25, 2017 14:13	October 25, 2017 14:13
1	ibm.com.br	October 25, 2017 14:13	October 25, 2017 14:13

3 white lists

Figura 33 - Dashboard de White List do Estudo de Caso Fonte: Próprio autor

Phalanxs Admin

Dashboard admin@admin.com Log out

**List of Users**

NAVIGATION  
Products  
Search results  
Tags  
**Users**  
White lists

Dashboard / Users

List Add new Export Start Search Add filter Selected items

Filter Refresh Export found Users

Name	Email	Sign in count	Current sign in at
Admin	admin@admin.com	1	October 25, 2017 14:20

1 user

Figura 34 - Dashboard de Usuários do Estudo de Caso Fonte: Próprio autor

Phalanxs Admin

Dashboard admin@admin.com Log out

**NAVIGATION**

- Products
- Search results**
- Tags
- Users
- White lists

Signed in successfully.

Dashboard / Search results

List Export Start Search Add filter Selected items

Filter Refresh

Export found Search results

Title	Link	Relevance	Occurrence	From	Created at	...
Pós UVA em Ensino da Arte com início em... - ...	https://pt-br.facebook.com/velgadealmeida/po...	1	1	Google	October 25, 2017 18:40	...
Notícias sobre Microsoft empresa -site:	/search?q=Microsoft+empresa+-site:&num=80...	1	1	Google	October 25, 2017 18:40	...
Microsoft - Wikipedia, la enciclopedia libre	https://es.wikipedia.org/wiki/Microsoft&sa=U&...	2	1	Google	October 25, 2017 18:40	...
IBM é a empresa de mais sucesso em uma tec...	https://conteudo.startse.com.br/corporate/feli...	2	1	Google	October 25, 2017 18:40	...
Microsoft demitirá milhares de funcionários em...	http://epocanegocios.globo.com/Empresa/noti...	3	1	Google	October 25, 2017 18:40	...
IBM Sata de imprensa - Marcelo Porto - Presid...	https://www.03.ibm.com/press/br/pt/biograph...	3	1	Google	October 25, 2017 18:40	...
Galeria de UVA El Paraíso / EDU - Empresa de...	https://www.archdaily.com.br/br/78897/uva-e...	3	1	Google	October 25, 2017 18:40	...
Empresa e Instituciones - Universidad de Vall...	http://www.uva.es/export/sites/uva/5.empresa...	4	1	Google	October 25, 2017 18:40	...
Microsoft lucra US\$ 5,2 bilhões em um trimestre...	https://adrenaline.uol.com.br/2017/01/27/4804...	5	1	Google	October 25, 2017 18:40	...
Sobre - UVA - Pedagogia é na UVA	http://estudenauba.com.br/sobre&sa=U&sqi=2...	5	1	Google	October 25, 2017 18:40	...
IBM em Portugal - Portugal	https://www.ibm.com/ibm/pt/pt/8sa=U&sqi=2...	5	2	Google	October 25, 2017 18:40	...
Empresa fabrica vinhos sem uva, por meio de...	http://www.inteligencia.com.br/empresa-fabri...	5	1	Google	October 25, 2017 18:40	...
Leonardo Almeida Veiga Perfil   Facebook	https://pt-br.facebook.com/public/Leonardo-Al...	6	7	Google	October 25, 2017 18:40	...
Empresa Festa da Uva teve prejuízo de quase ...	https://www.youtube.com/watch?v=0fR0dMn...	6	2	Google	October 25, 2017 18:40	...
Curso de Psicologia - UVA   Catho Educação	https://www.catho.com.br/educacao/curso/psi...	6	1	Google	October 25, 2017 18:40	...
Imagens de UVA empresa -site:	/search?q=UVA+empresa+-site:&num=80&saf...	7	4	Google	October 25, 2017 18:40	...
Ensino superior a distância vira modelo de suc...	http://odia.ig.com.br/noticia/educacao/2015-0...	7	2	Google	October 25, 2017 18:40	...

Figura 35 - Dashboard de Resultado de Busca do Estudo de Caso Fonte: Próprio autor

Sidekiq  ativo Painel Ocupados Filas Tentativas Agendados Morta

0 Processados	0 Falhas	5 Ocupados	0 Na fila	0 Tentativas	0 Agendados	0 Morta
---------------	----------	------------	-----------	--------------	-------------	---------

Processos

Name	Iniciados	Threads	Ocupados	
aebfd64cd326:1 app Filas: phalanx_default	há 4 horas	27	5	<b>Quiet</b> <b>Stop</b>

Tarefas

Process	TID	JID	Fila	Tarefa	Argumentos	Iniciados
aebfd64cd326:1:03403912017a	vo2ps	bab80166ef69f3d209d730db	phalanx_default	GoogleSearcherJob	"Microsoft empresa"	há 27 segundos
aebfd64cd326:1:03403912017a	vo4fg	fc320a173ea474940aed79be	phalanx_default	GoogleSearcherJob	"UVA empresa ensino"	há 27 segundos
aebfd64cd326:1:03403912017a	vo5a0	24127fdcf6e2ff11bfd9a323	phalanx_default	GoogleSearcherJob	"IBM empresa"	há 27 segundos
aebfd64cd326:1:03403912017a	vo27g	575e09dc0fb02054032ca2f	phalanx_default	GoogleSearcherJob	"UVA empresa"	há 27 segundos
aebfd64cd326:1:03403912017a	vo7i0	468df16d90d2d925243100b8	phalanx_default	GoogleSearcherJob	"UVA ensino"	há 27 segundos

Figura 36 - Dashboard de Filas do Sidekiq do Estudo de Caso Fonte: Próprio autor

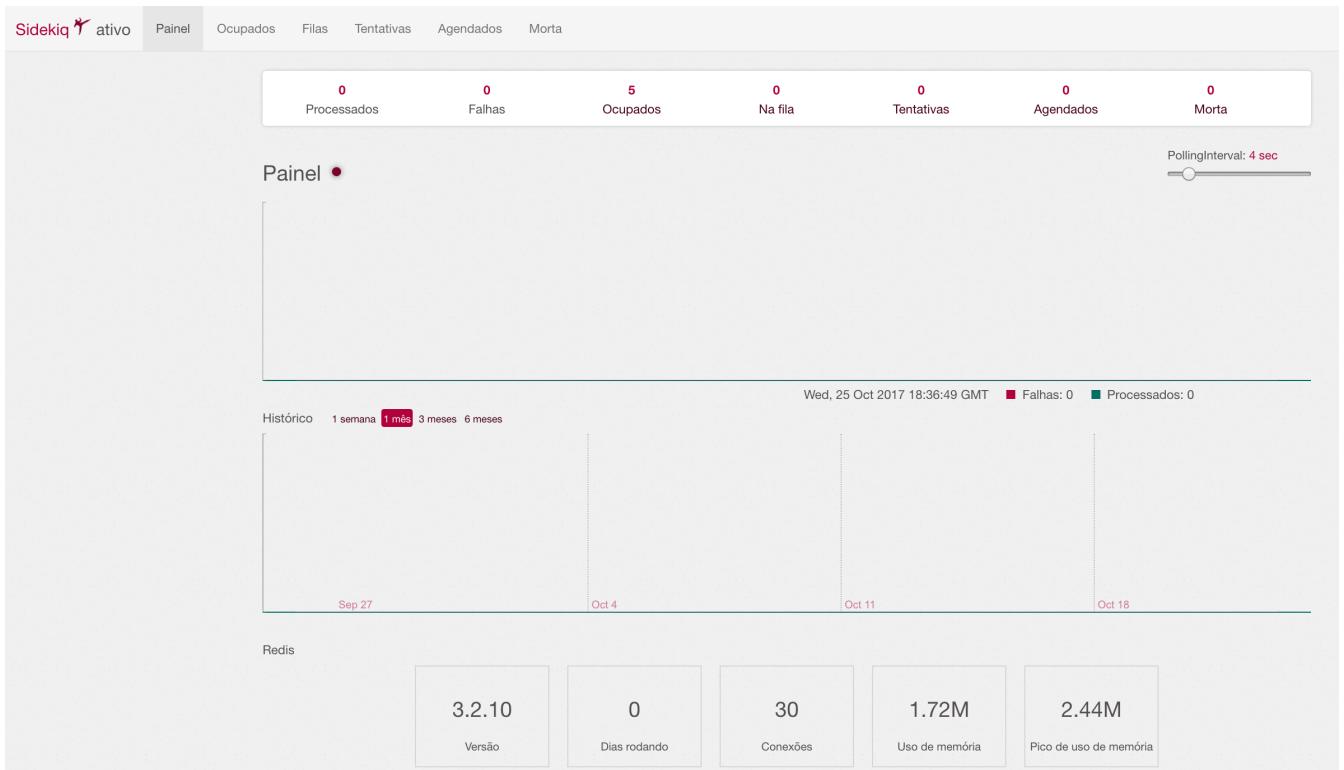


Figura 37 - Dashboard de Histórico de busca do Sidekiq do Estudo de Caso Fonte: Próprio autor

## 9 INFRAESTRUTURA

O estudo de caso está rodando em duas infraestruturas para poder comprovar a portabilidade de código efetuada pelo Docker.

Durante o desenvolvimento do estudo de caso conheci o projeto do RaspberryPi 3; Como se trata de um projeto que visa a implantação de conhecimentos de computação para crianças e pessoas sem possibilidades de acesso à computadores, sendo de baixo custo e com poder razoável de processamento; vi que era viável criar o projeto nesta arquitetura com o Docker, visto que esta placa é um pequeno computador.

Usei esta arquitetura para poder comprovar a portabilidade do docker e melhorar o meu estudo de caso, com a possibilidade de execução em infraestrutura própria e distribuída; Com o intuito de aprendizado sobre cluster e de uma nova arquitetura, pois queria desafios de aprendizado e não somente fazer deploy do meu cluster em uma infra-estrutura de terceiros como a AWS® ou a Digital Ocean®.

Para a orquestração dos containers estou usando o docker swarm, que é um orquestrador nativo da plataforma Docker. Pode ser usado outros orquestradores como o Kubernetes®, Vagrant® e Mesos OS®, sendo que este último não é suportado para a arquitetura ARM. A implementação do projeto nesses outros orquestradores ficará para melhorias futuras.

O projeto roda em arquitetura X64 para Linux e Mac e em uma arquitetura ARM para placas de RaspberryPi 3 usadas como infraestrutura própria ao projeto com a aplicação de cluster.

As placas utilizadas nesse projeto possuem as seguintes características:

- Modelo: Raspberry Pi 3 Model B;
- Sistema Operacional: Raspbian Debian 8;
- System embarcado usado: Broadcom BCM2837;
- CPU: 1.2 GHz 64/32-bit quad-core ARM Cortex-A53;
- Memória: 1 GB LPDDR2 RAM de 900 MHz;
- Armazenamento: MicroSDHC slot;
- Alimentação: 1.5 W.

Possuo dois arquivos de configurações do docker-compose e docker file adaptados para cada arquitetura (X64 e ARM). Foi necessário fazer essas adaptações devido a diferença de arquiteturas de processadores que executarão os códigos.

Arquivos das Arquiteturas:

X64:

Docker file: Apêndice 13.1

Docker-compose: Apêndice 13.2

ARM:

Docker file: Apêndice 13.3

Docker-compose versao 2 : Apêndice 13.4

Docker-compose versao 3 : Apêndice 13.5

Para esta infraestrutura, fiz utilização de um sistema operacional utilizado em um blog de um Docker Captain [39], que criou uma imagem de um sistema Linux Debian 8; usei o software Etcher [40] para poder inserir o sistema no cartão SD, e utilizá-lo no RaspberryPi.

## 10 CONCLUSÃO

Faço como conclusão desta obra que a computação em nuvem com o paradigma de serviço utilizando a plataforma Docker é de grande vantagem e proveito para a equipe desenvolvimento de software, quando para a equipe de infraestrutura, visto que visa aproximar ambas as equipes e ampliar a gama de conhecimento de ambas as equipes.

Vejo como vantagem este modelo de computação para *DataCenters*, visto que é de grande valia para escalonamento, provisionamento, criação e deleção de serviços, com foco cada vez mais na aplicação em microserviços e não mais tão focado na infraestrutura em si, visto que a mesma uma vez criada, pode ser replicada (Download) em qualquer host que possua o cliente do Docker.

Não fiz verificações e testes na versão EE desta plataforma para poder argumentar que este seja melhor do que o modelo CE, porém em estudo desta obra, verifiquei vantagens do modelo EE.

Visto que esta tecnologia possui grande aplicabilidade, versatilidade e gama de atuação, além de já ser utilizada por algumas grandes empresas a mesma me impressionou bastante no decorrer desta obra; porém recomendo que haja conhecimentos prévios sobre Linux.

A tecnologia em primeiro contato não é de fácil aprendizado, com uma curva de conhecimento bastante grande, exigindo bastante persistência e vontade de aprendizado de quem for fazer uso da mesma.

## 11 TRABALHOS FUTUROS

Como melhoria futura, proponho a adaptação do projeto para a orquestração em outros softwares orquestradores, como: Kubernet, Vagrant e Mesos OS.

A utilização destes orquestradores, resultará em adaptações feitas no projeto, pois cada um possui linguagem e sintaxe própria de utilização.

Isto será de grande aprendizado para ampliar conhecimentos e a utilização de novas tecnologias.

Proponho também como melhoria a utilização de deploy a partir de *CI (contínuos integrations)*. Estes softwares permitem fazer gestão de governança e todo o pipeline de deploy de uma aplicação de forma segura e com todos os registros e técnicas de governança necessárias para colocar uma aplicação em produção seguindo os melhores parâmetros e métodos estipulados por cada equipe, empresa. Não o fiz, pois estava em caráter experimental e de estudos, então criei scripts em *shell* para poder fazer o *build* da imagem do Docker-compose, *seed* do banco de dados e *scaffold* dos estáticos de forma que pudesse replicar de forma rápida com o GitHub® para todos os hosts utilizados em infraestrutura própria.

Proponho a verificação e utilização de outras plataformas web (*PaaS*) para poder fazer *build* da aplicação, como o Heroku®, que é possível fazer o *deploy*, gerenciamento e escalonamento da aplicação para outros *IaaS*. Não tenho por intuito prender ou indicar quaisquer outras plataformas ou outros métodos nesta obra, somente fiz informação em caráter de conhecimento de outros métodos e plataformas que tive contato no decorrer desta obra.

## **12 REFERÊNCIAS**

- [1] “The NIST Definition of Cloud Computing”. Disponível em <<http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>>. Acessado em 24/11/2016
- [2] História da computação em nuvens. Disponível em <[http://www.dsc.ufcg.edu.br/~pet/jornal/agosto2012/materias/historia\\_da\\_computacao.html](http://www.dsc.ufcg.edu.br/~pet/jornal/agosto2012/materias/historia_da_computacao.html)>. Acessado em 18/11/2016
- [3] NOGUEIRA, MATHEUS CADORI1; PEZZI, DANIEL DA CUNHA (2010) “A Computação Agora é nas Nuvens” Universidade de Cruz Alta (UNICRUZ) – Cruz Alta, RS – Brasil.
- [4] AULBACH, STEFAN; JACOBS, DEAN; KEMPER, ALFONS; et al. (2009) “A Comparison of Flexible Schemas for Software as a Service.” 35th SIGMOD - International Conference on Management of Data, 2009.
- [5] TAURION, C. (2009) “Cloud Computing: computação em nuvem: transformando o mundo da tecnologia da informação”, Editora Brasport: Rio de Janeiro, Brasil.
- [6] VERAS, MANOEL.(2012) “Cloud Computing: Nova Arquitetura da TI”. Editora Brasport:Rio de Janeiro, Brasil.
- [8] SOUZA, FLÁVIO R. C.; MOREIRA, LEONARDO O.; MACHADO, JAVAM C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. ERCEMAP 2009.
- [9] “Saiba como a virtualização por container revolucionou a infraestrutura de TI. Parte 2”. Disponível em <<https://www.3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti-part2/>>. Acessado em 03/05/2017
- [10] “Saiba como a virtualização por container revolucionou a infraestrutura de TI. Parte 1”. Disponível em <<https://www.3way.com.br/saiba-como-a-virtualizacao-por-container-mudou-a-infraestrutura-de-ti/>>. Acessado em 03/05/2017

[11] “Saiba como a virtualização por container revolucionou a infraestrutura de ti! Parte 3”. Disponível em <<https://www.3way.com.br/gerenciando-containers-usando-kubernetes/>>. Acessado em 03/05/2017

[12] “O que é Container?”. Disponível em <<http://www.mundodocker.com.br/o-que-e-container/>>. Acessado em 04/05/2017

[13] “10 things to avoid in docker containers”. Disponível em: <<https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/>>. Acessado em 25/10/2017

[14] “Docker Engine”. Disponível em <<https://docs.docker.com/engine/docker-overview/#docker-engine>> Acessado em 25/10/2017

[15] “GOMES, RAFAEL “Docker para Desenvolvedores”. Editora 9 Bravos: Rio de Janeiro, Brasil. Disponível em <<https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/porque.md>> Acessado em 25/10/2017

[16] “Docker for Mac”. Disponível em <<https://www.docker.com/docker-mac>> Acessado em 26/11/2017.

[17] “Best practices for writing Dockerfiles”. Disponível em <[https://docs.docker.com/engine/userguide/eng-image/dockerfile\\_best-practices/](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/)> Acessado em 10/06/2017

[18] “Docker glossary. Compose”. Disponível em <<https://docs.docker.com/glossary/?term=Compose>> Acessado em 15/06/2017

[19] “Docker glossary. Dockerfile”. Disponível em <<https://docs.docker.com/glossary/?term=Dockerfile>> Acessado em 15/06/2017

[21] “Docker para Desenvolvedores - Porque”. Disponível em <<https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/porque.md>> Acessado em 16/06/2017

[22] “Software Versioning”.Disponível em <[https://en.wikipedia.org/wiki/Software\\_versioning](https://en.wikipedia.org/wiki/Software_versioning)> Acessado em 16/06/2017

[23] “Docker Container”. Disponível em <<https://docs.docker.com/glossary/?term=container>> Acessado em 16/06/2017

[24] “Docker commands”. Disponível em <<https://docs.docker.com/engine/reference/commandline/docker/>> Acessado em 10/07/2017

[25] “Docker Hub Portainer”. Disponível em <<https://hub.docker.com/r/portainer/portainer/>> Acessado em 10/07/2017

[26] “Docker Swarm”. Disponível em <<https://docs.docker.com/glossary/?term=Docker%20Swarm>> Acessado em 10/07/2017

[27] “Docker Swarm Visualizer”. Disponível em <<https://github.com/dockersamples/docker-swarm-visualizer>> Acessado em 10/07/2017

[28] “Sobre Docker Labs”. Disponível em <<http://training.play-with-docker.com/about>> Acessado em 20/07/2017

[29] “Docker Tutorials em Labs”. Disponível em <<https://github.com/docker/labs>> Acessado em 20/07/2017

[30] “Play with Docker Classroom”. Disponível em <<http://training.play-with-docker.com/>> Acessado em 20/07/2017

[32] “Docker Conf 2017 – Austin, TX”. Disponível em <<https://2017.dockercon.com/>> Acessado em 05/08/2017

[33] “Docker Store”. Disponível em <<https://store.docker.com/>> Acessado em 07/08/2017

[34] “Docker Store - Docker Enterprise Edition for AWS”. Disponível em <<https://store.docker.com/editions/enterprise/docker-ee-aws>> Acessado em 07/08/2017

[35] “Docker and IBM Offerings”.Disponível em <<https://www.docker.com/ibm>> Acessado em 15/08/2017

[36] “Deploy scalable Docker application to IBM Bluemix” .Disponível em <<https://www.ibm.com/blogs/bluemix/2017/02/deploy-scalable-docker-application/>> Acessado em 15/08/2017

[38] “12 Factors App”. Disponível em <[https://12factor.net/pt\\_br](https://12factor.net/pt_br)> Acessado em 17/08/2017

[39] “Blog Hypriot”. Disponível em <<https://blog.hypriot.com/downloads>> Acessado em 18/08/2017

[40] “Etcher”. Disponível em <<https://etcher.io>> Acessado em 18/08/2017

Cloud Computing. Disponível em <[https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)>. Acessado em 20/11/2016.

O que é cloud computing. Disponível em <<http://www.infowester.com/cloudcomputing.php>>. Acessado em 20/11/2016.

Conceitos de computação em nuvem. Disponível em <<http://ftp.unipar.br/~seinpar/2013/artigos/Rogerio%20Schueroff%20Vandresen.pdf>>. Acessado em 24/11/2016

## 13 APÊNDICE

### Apêndice 13.1 – Dockerfile - X64

```
FROM ubuntu:12.04
```

```
FROM ruby:2.3.3
```

```
MAINTAINER Thiago Soares <thiagosoarescruz0@gmail.com>
```

```
# Install Build essentials and MySQL client
```

```
RUN apt-get update -qq && apt-get install -y sudo build-essential \
    libpq-dev nodejs-legacy mysql-client \
    libssl-dev apt-utils nodejs mysql-client && \
    sudo apt-get autoremove -y && \
    sudo rm -rf /var/lib/apt/lists/*
```

```
# Set some config
```

```
ENV RAILS_LOG_TO_STDOUT true
```

```
ENV RAILS_ENV=development
```

```
#Mkdir
```

```
RUN mkdir -p /home/app
```

```
#Workdir
```

```
WORKDIR /home/app/
```

```
#Add Docker path
```

```
ADD docker /home/app/
```

```
WORKDIR /home/app
```

```
#Add gems
```

```
ADD Gemfile /home/app/Gemfile
```

```
ADD Gemfile.lock /home/app/Gemfile.lock
```

```
#Add sidekiq pid
ADD sidekiq.pid /home/app/tmp/pids/

#Run bundle
RUN bundle install

#Add the Rails app
ADD . /home/app

#Create user and group
RUN groupadd --gid 9999 app && \
    useradd --uid 9999 --gid app app && \
    chown -R app:app /home/app

#Expose app port
EXPOSE 80 300 9000

#Save timestamp of image building
RUN date -u > BUILD_TIME
```

## Apêndice 13.2 – Docker-compose versão 2 - X64

```
version: '2'
services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: phalanx-development
      MYSQL_USER: root
      MYSQL_PASSWORD: password
    ports:
      - "3307:3306"
    volumes:
      - "db-data:/var/lib/mysql"

  redis:
    image: redis:3.2-alpine
    restart: always

  app: &app_base
    build: .
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    restart: always
    volumes:
      - ./www/phalanx/app
    environment:
      REDIS_SIDEKIQ_URL: redis://redis:6379/0
      REDIS_CABLE_URL: redis://redis:6379/1
      DB_HOST: db
      DB_USER: root
      DB_NAME: phalanx-development
      DB_PASSWORD: password
```

```
ports:  
  - "3001:3000"
```

```
depends_on:  
  - db  
  - redis
```

```
links:  
  - redis  
  - db
```

```
worker:
```

```
<<: *app_base  
command: bundle exec sidekiq  
ports: []  
restart: always  
depends_on:  
  - app
```

```
ui:
```

```
image: portainer/portainer  
restart: always  
volumes:  
  - '/var/run/docker.sock:/var/run/docker.sock'  
expose:  
  - 9000  
ports:  
  - 3002:9000
```

```
volumes:
```

```
db-data:
```

### Apêndice 13.3 – Dockerfile – ARM

```
FROM resin/rpi-raspbian:jessie
```

```
MAINTAINER Thiago Soares <thiagosoarescruz0@gmail.com>
```

```
#Install Build essentials e MySQL client
```

```
RUN apt-get update -qq && apt-get install -y mysql-client \  
    build-essential ca-certificates curl build-essential \  
    libpq-dev nodejs nodejs-legacy mysql-server libmysqlclient-dev && \  
    sudo apt-get autoremove -y && \  
    sudo rm -rf /var/lib/apt/lists/*
```

```
#Mkdir
```

```
RUN mkdir -p /home/app
```

```
#Workdir
```

```
WORKDIR /home/app/
```

```
#Add Docker path
```

```
ADD docker /home/app/
```

```
#Add sidekiq pid
```

```
ADD sidekiq.pid /home/app/tmp/pids/
```

```
#ADD gems
```

```
ADD Gemfile /home/app/Gemfile
```

```
ADD Gemfile.lock /home/app/Gemfile.lock
```

```
#Install RVM, Ruby, and Bundler
```

```
RUN \curl -sSL https://rvm.io/mpapis.asc | gpg --import -
```

```
RUN \curl -L http://get.rvm.io | bash -s stable
```

```
RUN /bin/bash -l -c "source /etc/profile.d/rvm.sh"
```

```
RUN /bin/bash -l -c "rvm install 2.3.3"
```

```
RUN /bin/bash -l -c "gem install bundle --no-ri --no-rdoc"
```

```
#Workdir
```

```
#WORKDIR /home/app/
```

```
#Run bundle
```

```
RUN /bin/bash -l -c "bundle install"
```

```
# Add the Rails app
```

```
ADD . /home/app
```

```
#Create user and group
```

```
RUN groupadd --gid 9999 app
```

```
RUN useradd --uid 9999 --gid app app
```

```
RUN chown -R app:app /home/app
```

```
#Save timestamp of image building
```

```
RUN date -u > BUILD_TIME
```

```
# Expose app port
```

```
EXPOSE 80 3000
```

## Apêndice 13.4 – Docker-compose versão 2 - ARM

```
version: '2'
services:
  db:
    image: hypriot/rpi-mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: phalanx-development
      MYSQL_USER: root
      MYSQL_PASSWORD: password

    ports:
      - "3307:3306"
    volumes:
      - "db-data:/var/lib/mysql"

  redis:
    image: hypriot/rpi-redis

  app: &app_base
    build: .
    command: bash -lc 'bundler exec rails s -b 0.0.0.0'
    volumes:
      - ./www/phalanx/app
    environment:
      REDIS_SIDEKIQ_URL: redis://redis:6379/0
      REDIS_CABLE_URL: redis://redis:6379/1
      DB_HOST: db
      DB_USER: root
      DB_NAME: phalanx-development
      DB_PASSWORD: password

    ports:
      - "3001:3000"
```

```
depends_on:
  - db
  - redis

links:
  - redis
  - db

worker:
<<: *app_base
#command: bundle exec sidekiq
  command: bash -lc 'bundle exec sidekiq'
  ports: []
depends_on:
  - app

ui:
  image: portainer/portainer
  restart: always
  volumes:
    - '/var/run/docker.sock:/var/run/docker.sock'
  expose:
    - 9000
  ports:
    - 3002:9000

volumes:
  db-data:
```

## Apêndice 13.5 – Docker-compose versão 3 - ARM

```
version: "3"

services:
  db:
    image: hypriot/rpi-mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: phalanx-development
      MYSQL_USER: root
      MYSQL_PASSWORD: password
    ports:
      - "3307:3306"
    volumes:
      - "db-data:/var/lib/mysql"
    networks:
      - phalanx_app
  deploy:
    placement:
      constraints: [node.role == manager]

  redis:
    image: hypriot/rpi-redis
    ports:
      - "6379:6379"
    volumes:
      - "/home/docker/data:/data"
    networks:
      - phalanx_app
  deploy:
    placement:
      constraints: [node.role == worker]
```

```
app: &app_base

image: phalanx_app
command: bash -lc 'bundler exec rails s -b 0.0.0.0'
volumes:
  - ./www/phalanx/app
environment:
  REDIS_SIDEKIQ_URL: redis://redis:6379/0
  REDIS_CABLE_URL: redis://redis:6379/1
  DB_HOST: db
  DB_USER: root
  DB_NAME: phalanx-development
  DB_PASSWORD: password
ports:
  - 3001:3000
depends_on:
  - db
  - redis
networks:
  - phalanx_app

#service deployment
deploy:
  mode: replicated
  replicas: 1
  labels: [APP=PHALANX]
#service resource management
resources:
  #Hard limit - Docker does not allow to allocate more
  limits:
    cpus: '0.25'
    memory: 512M
  #Soft limit - Docker makes best effort to return to it
  reservations:
```

```
cpus: '0.25'  
memory: 256M  
  
#service restart policy  
restart_policy:  
    condition: on-failure  
    delay: 5s  
    max_attempts: 10  
    window: 120s  
  
#service update configuration  
update_config:  
    parallelism: 1  
    delay: 10s  
    failure_action: continue  
    monitor: 60s  
    max_failure_ratio: 0.3  
  
#placement constraint - in this case on 'worker' nodes only  
placement:  
    constraints: [node.role == manager]
```

```
dockerui:  
    image: portainer/portainer  
    volumes:  
        - '/var/run/docker.sock:/var/run/docker.sock'  
    ports:  
        - 8080:9000  
    networks:  
        - phalanx_app  
deploy:  
    placement:  
        constraints: [node.role == worker]
```

```
worker:  
    <<: *app_base  
    image: phalanx_worker
```

```
command: bash -lc 'bundler exec sidekiq'
```

```
ports:
```

```
  - 3003:3003
```

```
networks:
```

```
  phalanx_app:
```

```
    aliases:
```

```
      - workers
```

```
depends_on:
```

```
  - app
```

```
deploy:
```

```
  placement:
```

```
    constraints: [node.role == worker]
```

```
viz:
```

```
  image: alexellis2/visualizer-arm
```

```
  ports:
```

```
    - 8081:8081
```

```
  deploy:
```

```
    placement:
```

```
      constraints: [node.role == worker]
```

```
  volumes:
```

```
    - '/var/run/docker.sock:/var/run/docker.sock'
```

```
networks:
```

```
  phalanx_app:
```

```
volumes:
```

```
  db-data:
```