**­­­­UNIVERSIDADE VEIGA DE ALMEIDA – UVA**

**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**DOCKER: UM CANIVETE SUÍÇO**

**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 Lemos

**DOCKER: UM CANIVETE SUÍÇO**

**RIO DE JANEIRO**

**2017**

image003

**UNIVERSIDADE VEIGA DE ALMEIDA - UVA**

**BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO**

**THIAGO SOARES DA CRUZ**

**DOCKER: UM CANIVETE SUÍÇO**

Monografia apresentada como requisito parcial à conclusão do curso em Bacharel em Ciência da Computação.

APROVADA EM:

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

BANCA EXAMINADORA:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PROF.**

**ORIENTADOR**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PROF.**

**CO-ORIENTADOR**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PROF.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PROF .**

**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.*

“Conhecer, é poder”

Fonte: Francis Bacon

**RESUMO**

Palavras-Chave:

**ABSTRACT**

**LISTA DE ILUSTRAÇÕES**

Figura 1: Clound 14

Figura 2: Joseph Carl 15

Figura 3: John McCarthy 15

Figura 4: Ramnath Chellappa 16

Figura 5: Modelos de Implementação 17

Figura 6: Nuvem pública 18

Figura 7: Nuvem Privada 19

Figura 8: Nuvem Comunitária 20

Figura 9: Modelos de Implementação 21

Figura 10: Modelos de Serviço 24

Figura 11 Hypervisor Hospedado 25

Figura 12 - Evolução do Hypervisor 26

Figura 13 - Virtualização por Container 26

Figura 14 - LXC Container 28

Figura 15 LXC vs KVM 29

Figura 16 - Play with docker 37

Figura 17 - Shell do Play with Docker. 38

**­­LISTA DE ABREVIATURAS E SIGLAS**

Paas – Plataforma as a Service – Plataforma como serviço

Deploy – Implementação de software em ambiente/maquina (QA — testes ou de Prod — produção)

QA – Quality Assurance (Ambiente de mensuração de Qualidade/Testes de novas implementações)

Downtime – Tempo de queda (instabilidade) de uma aplicação ao ser colocada em um ambiente via

Deploy.

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.

Dev – Ambiente local de desenvolvimento do programador/analista.

Prod – Ambiente de Produção, da qual o software é entregue ao cliente final.

OS – Operation System – Sistema Operacional.

Kernel – Núcleo do Sistema Operacional.

**SUMÁRIO**

[INTRODUÇÃO 12](#_Toc496267346)

[2 COMPUTAÇÃO EM NUVEM 14](#_Toc496267347)

[2.1 História 15](#_Toc496267348)

[2.2 Modelos de implantação 17](#_Toc496267349)

[2.2.1 Nuvem Pública 18](#_Toc496267350)

[2.2.2 Nuvem Privada 19](#_Toc496267351)

[2.2.3 Nuvem Comunitária 20](#_Toc496267352)

[2.2.3 Nuvem Híbrida 21](#_Toc496267353)

[2.3 Princípios da computação em nuvem 22](#_Toc496267354)

[2.4 Modelos de Serviços 24](#_Toc496267355)

[3 Container vs Virtualização 25](#_Toc496267356)

[3.1 LXC Containers 28](#_Toc496267357)

[4. Docker 30](#_Toc496267358)

[4.1 motivos para usar o Docker 32](#_Toc496267359)

[4.2 Docker Imagem 33](#_Toc496267360)

[4.2.1 Docker Hub 34](#_Toc496267361)

[4.3 Docker-Compose 35](#_Toc496267362)

[4.4 Play with Docker 37](#_Toc496267363)

[4.5 Comunidade 38](#_Toc496267364)

[5 Boas práticas de Construção da aplicação (Doze fatores) 39](#_Toc496267365)

[6 Softwares de Gerenciamento de Containers 41](#_Toc496267366)

[9 Plataformas de Orquestração 42](#_Toc496267367)

[12 Empregabilidade da técnica 43](#_Toc496267368)

[13 Estudo de Caso 44](#_Toc496267369)

[14 Infraestrutura 45](#_Toc496267370)

[19 Referências 49](#_Toc496267371)

[11 Anexos 50](#_Toc496267372)

INTRODUÇÃO

A computação em nuvem tem como enfoco 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], 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, utilizado 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 (esta que abalou os diversos fatores culturais e áreas da sociedade), processamento e o armazenamento. Os dados foram se tornando cada vez mais importante, onde ao passar das épocas, foi-se tornando visível o avanço quanto a necessidade de dados e consequentemente a esta visualização, se foi percebendo um acumulo cada vez maior de informações.

A muito invisível para os usuários como seus aplicativos processam os seus recursos (meta-dados), 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 downtime (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 micro-serviços.

Este tipo de programação separa o 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 do 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 micro-serviço, 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 nesse trabalho, porém a mesma será meramente aplicável ao trabalho.

Este trabalho se dividirá além da introdução em outros capítulos: 2 Computação em nuvem da qual irei fazer uma fundamentação histórica e teórica sobre a computação em nuvem, modelos de implantação, suas vantagens e desvantagens, seus modelos de serviços, a elasticidade para os seus serviços e pagamento pela utilização de recursos; além de argumentar sobre esses modelos e falar sobre falhas e suas tolerâncias.

No capítulo 3, irei apresentar o conceito de PaaS, plataforma como serviço e alguns outros conceitos relativos a computação em nuvem, falarei sobre modelos de programação descentralizada e os conceitos de grid, cluster e docker implementados para esse tipo de plataforma e modelo de programação.

No capítulo 4, irei falar sobre alguns softwares de deploy (implementação) e os conceitos relativos a essa nova arquitetura de PaaS, me fundamentarei em alguns como: o Tsuru, Vagran, Kubernet; irei fazer fundamentação teórica sobre cada plataforma, além de fazer uma breve comparação (no capítulo 4).

No capítulo 5, irei falar sobre sistemas escalares, ou seja, sistemas operacionais que já fazem a implementação de escalabilidade de forma nativa como se fosse um verdadeiro PaaS, esses sistemas foram criados para este tipo de arquitetura;

No capítulo 6, irei fazer uma comparação teórica com pontos positivos e negativos sobre as PaaS abordadas; Irei fazer esta comparação somente no capítulo 6 e não no capítulo 4 pois faço necessário informar sobre sistemas escalares e irei fazer comparação desses com os PaaS.

No capítlulo 7, irei abordar e fundamentar a teoria sobre programação em micro-serviço, boas práticas desse modelo de programação para o novo modelo de arquitetura descentralizado e escalável.

No capítulo 8, irei falar sobre o estudo de casos abordado no trabalho, farei fundamentação teórica para o motivo de meu estudo e trabalho, além de falar sobre infraestrutura relativa à este tipo de arquitetura e aplicabilidade deste modelo de computação.

No capítulo 9, 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.

2 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).

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: Clound Fonte: http://www.synergixtech.com/wp-content/uploads/2016/09/Cloud-Computing-Benefits.png

2.1 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.

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 à 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.

2.2 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.

2.2.1 Nuvem Pública



Figura 6: Nuvem pública Fonte: 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.

2.2.2 Nuvem Privada

Figura 7: Nuvem Privada Fonte: 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.

2.2.3 Nuvem Comunitária

Figura 8: Nuvem Comunitária Fonte: https://lh4.googleusercontent.com/NWf67CzmfbXLVsj60ZRyC-eX-UcdAt3ITRZcgedyN4dBbGU0BOWdCSNdtuqz9DxZ4fHNC6GLnUlreeoRX\_\_8c07l61YMMoY3zxKJvMkbFfx92vjjDYLm1ai2STm0h4XQfA

Uma infraestrutura de nuvem comunitária é contratada por um grupo de organizações em conjunto ou programas que compartilham necessidades especificas, 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.

2.2.3 Nuvem Híbrida



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.

2.3 Princípios da computação em nuvem

Segundo NIST (2011), 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, mais 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 à falhas, da qual em último caso de não poder ser tolerante à falhas e 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.

2.4 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 possiblidade 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).
* 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).
* 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).



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

3 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, mais 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 pode ser tem um alto nível de provisionamento, podendo ser replicado sem menores problemas, simplificando muito a implementação em diferentes máquinas e ambientes de homologação (QA – Quality Assurance, Staging, Produção); o container pode ser baixado em qualquer lugar, e máquina hospedeira 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.

3.1 LXC Containers

Conforme citação do Rogério dos Anjos (em [Linux](http://3way.com.br/category/linux/), [Novidades](http://3way.com.br/category/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”.[[1]](#footnote-1)1

Conforme citação de [Cristiano Diedrich](http://www.mundodocker.com.br/author/cristiano/) – “o projeto do LXC, trazia as seguintes fases: LXC, chroot com esteroides. 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.”[[2]](#footnote-2)1

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](http://3way.com.br/category/linux/), [Novidades](http://3way.com.br/category/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.”[[3]](#footnote-3)2



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

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 carácter open-source.



Figura 15 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[[4]](#footnote-4)2 - existem alguns passos à serem seguidos para execução com cantainers:

* Primeiro: Containers são imutáveis - O OS, 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. Instanciado em minutos, é necessário iniciar um novo container em segundos.

4. 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.

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.[[5]](#footnote-5)3

**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\*.

Você pode instalar via brew cask com o comando abaixo:

brew cask install docker

ou é possível instalar o cliente pelo próprio site do Docker[[6]](#footnote-6)4

4.1 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.[[7]](#footnote-7)5

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 (do sistema), system-calls e etc.
* Agregabilidade e Previsibilidade
  + Agregação de serviços para elicitar a aplicação de previsibilidade de execução e conclusão de execução.

4.2 Docker IMAGEM

A imagem do docker é um pacote da aplicação, que acabou de ser criada, “buildada” 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 em um sistema de camadas 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. [[8]](#footnote-8)6

<imagem de imagem do docker >

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 (Libs) 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.

4.2.1 DockerHUB

O Docker disponibiliza uma nuvem pública para compartilhamento de imagens, que podem ser utilizados para viabilizar customizações para 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 da construção da imagem e da tag.

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

<inserir comando para baixar a imagem>

<inserir imagem do Docker hub>

4.3 Docker-Compose

Existe uma ferramenta do Docker que é o Docker-compose, da qual é possível escrever um único arquivo em formato shell e o mesmo será a imagem para a criação dos containers.

Essa técnica possui algumas aplicações:

* Isolamento do sistema operacional
  + É possível utilizar um sistema operacional para a imagem diferente do sistema operacional nativo do host, da qual o Docker está sendo executado — Em versões mais antigas é necessário a instalação do Boot2docker — que faria essa camada de abstração do Docker ao Sistema Operacional. Porém já foi corrigido em versões prévias do Docker.
* Controle de arquivos
  + Permite fazer um controle de pastas e arquivos que serão interessantes e/ou necessários à aplicação acessar para manter a execução. Como por exemplo diretório de arquivos próprios da aplicação.

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

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 é 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 “buildada” 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 “biuldados” e executados em separado. Por exemplo o banco de dados em MySQL ele presta serviços para a aplicação, a mesma consegue ser construída “biuldada” e ficar “up” sem o banco; não vai ter sua completa utilização e execução sem o mesmo, mais é possível.

Todos os serviços tem que estar vinculados no arquivo “.yml”, fazendo referência à imagem que foi criada pelo Dockerfile. A imagem é construída “biuldada” antes da execução do container que faz referência a ela, e 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.

4.3 Docker File

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.

No arquivo de Docker file é possível fazer especificações de sistema operacional que será utilizado para construir a imagem; bem como as bibliotecas que serão utilizadas para construir a imagem.

Instalações que permanecerão para a imagem também podem ser inseridas no momento de construção do Docker file; No estudo de caso para a arquitetura ARM eu faço a instalação do RVM, Ruby e bundler das dependências do projeto. Estas serão necessárias a execução.

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 não estava acostumado com a mesma, isto implicou em criar e recriar este arquivo de Dockerfile 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 será reiniciado.

De acordo com Rafael Benevides – Diretor de Experiência de Desenvolvimento da Red Hat[[9]](#footnote-9)2 – “Containers são descartáveis”.

4.4 PLAY WITH DOCKER

O Docker possui uma engine de testes do seu próprio labortório, daqual é possível acessar através do link: <http://labs.play-with-docker.com>.

Essa engine consiste em uma máquina virtual com Alphine 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. [[10]](#footnote-10)7

Maiores informações podem ser obtidas no github do projeto[[11]](#footnote-11)8; Este projeto também é utilizado para treinamentos do Docker.



Figura 16 - Play with docker - Fonte: http://labs.play-with-docker.com/



Figura 17 - Shell do Play with Docker. Fonte: http://host3.labs.play-with-docker.com/p/9fe5a73e-9d79-4b61-9c01-3365068d8bd2#9fe5a73e\_node2

O projeto conta com uma documentação que poder ser obtida no link: <http://training.play-with-docker.com/>

Outros treinamentos do docker podem ser obtidas através do link: <https://training.docker.com/>

4.5 coMUNIDADE

5 Boas práticas de Construção da aplicação (Doze fatores)

Esses doze fatores, são fatores para a melhor criação, mautenção, 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.[[12]](#footnote-12)9

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 e**ntre 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”[[13]](#footnote-13)10 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.

**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.

6 SOFTWARES DE GERENCIAMENTO DE CONTAINERS

Existem algumas Plataformas (PaaS) para gerenciamento de containers em ambientes (qa, staging e produção).

Essas plataformas tem a finalidade de melhorar o condicionamento de containers, de forma visual, para que não seja necessário conhecimento 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 é condicionamento dos containers elas melhoram 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.

Essas PaaS, citadas possuem integração com a AWS (a menos o Tsuru – o mesmo é baseado em Docker, restrito ao Data Center do grupo Globo).

9 Dificuldades técnicas encontradas

13 ESTUDO DE CASO

Como estudo de caso desta monografia, utilizarei um crawler de internet, desenvolvido na linguagem de programação Ruby.

Fiz a escolha desta linguagem de forma arbitrária, pois não tinha conhecimentos da linguagem e tomei como desafio aprender um pouco sobre uma nova linguagem e o paradigma da computação em serviço.

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

<Adicionar aqui modelo de documento de aplicação que está sendo escrito para o vod\_plataforma>

<Adicionar diagrama de caso de uso e de sequência >

O mesmo foi criado sobre o paradigma de programação por serviço, 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.

­­­

**<Documentação de SW a ser inserida>**

1. **Objetivo**
2. **Cenário Atual**
3. **Descrição do Projeto**
4. **Restrições (Funcionais, tecnológicas, operacionais e organizacionais)**
5. **Proposta de solução Tecnológica Escolhida.**

14 Infraestrutura

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

O projeto roda em arquitetura X86 para Linux e Mac e em uma arquitetura ARM para placas de RaspberryPi 3 usadas como infraestrutura própria ao projeto.

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

Arquivos das Arquiteturas:

X86:

Docker file: Anexo 1

Docker-compose: Anexo 2

ARM:

Docker file: Anexo 3

Docker-compose versao 2 : Anexo 4

Docker-compose versao 2 : Anexo 5

Durante o desenvolvimento do projeto conheci o projeto do RaspberryPi 3[[14]](#footnote-14)6; Como se trata de um projeto que visa a implantação de conhecimentos e possibilidades de acesso à computadores de baixo custo e com poder razoável de processamento. 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.

Para a orquestração dos containers estou usando o docker swarm, que é um orquestrador nativo da plataforma. Pode ser usado outros orquestradores como o Kubernet, Vagrant e Mesos OS. A utilização destes orquestradores ficará para melhorias futuras.

**15 CONCLUSAO**

**16 MELHORIAS FUTURAS**

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.

19 Referências

História da computação em nuvens. Disponível em <http://www.dsc.ufcg.edu.br/~pet/jornal/agosto2012/materias/historia\_da\_computacao.html>. Acessado em 18/11/2016

Clound Computing. Disponível em <https://en.wikipedia.org/wiki/Cloud\_computing>. Acessado em 20/11/2016.

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

Fundamentos de Computação Nuvem para Governos. Disponível em <http://www4.serpro.gov.br/wcge2010/artigos/Artigo-Fundamentos%20de%20Computacao%20Nuvem%20para%20Governos.pdf>. Acessado em 21/11/2016.

Clound computing estamos nas nuvens voando. Disponível em <http://ppgi.ci.ufpb.br/wp-content/uploads/palestra-cloud-computing.pdf>. Acessado em 21/11/2016

“The NIST Definition of Cloud Computing”. Disponível em <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf >. Acessado em 24/11/2016

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

Nogueira, Matheus Cadori1; PEZZI, Daniel da Cunha (2010) “A Computação Agora é nas Nuvens” Universidade de Cruz Alta (UNICRUZ) – Cruz Alta, RS – Brasil.

Veras, Manoel (2012) “Cloud Computing: Nova Arquitetura da TI”. Editora Brasport: Rio de Janeiro, Brasil.

Souza, Flávio R. C.; Moreira, Leonardo O.; Machado, Javam C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. ERCEMAPI 2009.

“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

“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

“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

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

“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

11 Anexos

Anexo 1 – Dockerfile - X86

FROM ubuntu:12.04

FROM ruby:2.3.3

MAINTAINER Thiago Soares <thiagosoarescruz0@gmail.com>

## Install Build essentials

RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs-legacy mysql-client \

libssl-dev apt-utils nodejs

# Install MySQL client

RUN apt-get update && \

apt-get -y install sudo

RUN useradd -m docker && echo "docker:docker" | chpasswd && adduser docker sudo

#USER docker

#RUN docker ALL = NOPASSWD: /sbin/poweroff, /sbin/start, /sbin/stop

# Install MySQL client

RUN sudo apt-get install -y mysql-client && \

sudo apt-get autoremove -y && \

sudo rm -rf /var/lib/apt/lists/\*

# Set some config

ENV RAILS\_LOG\_TO\_STDOUT true

# 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

Anexo 2 – Docker-compose versão 2 - X86

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:

Anexo 3 – Dockerfile – ARM

FROM resin/rpi-raspbian:jessie

MAINTAINER Thiago Soares <thiagosoarescruz0@gmail.com>

# Install MySQL client

RUN apt-get update -qq

# Install Build essentials

RUN apt-get update -yq

RUN 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"

#RUN /bin/bash -l -c "apt-get install -y rubygems"

#RUN /bin/bash -l -c "gem install rails"

# 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

Anexo 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: bundler exec sidekiq

command: bash -lc 'bundler 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:

Anexo 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

#restart: always

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:

1. 1 https://www.3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti-part2/ [↑](#footnote-ref-1)
2. 1 https://www.mundodocker.com.br/o-que-e-container/ [↑](#footnote-ref-2)
3. 2 https://www.3way.com.br/saiba-como-a-virtualizacao-por-container-revolucionou-a-infraestrutura-de-ti-part2/ [↑](#footnote-ref-3)
4. 2 https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/, tradução nossa [↑](#footnote-ref-4)
5. 3 https://docs.docker.com/engine/docker-overview/#docker-engine, tradução nossa [↑](#footnote-ref-5)
6. 4 https://www.docker.com/docker-mac [↑](#footnote-ref-6)
7. 5 https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/porque.md [↑](#footnote-ref-7)
8. 6 https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/porque.md - Dúvidas [↑](#footnote-ref-8)
9. 2 https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/, tradução nossa [↑](#footnote-ref-9)
10. 7 http://training.play-with-docker.com/about/ [↑](#footnote-ref-10)
11. 8 https://github.com/docker/labs [↑](#footnote-ref-11)
12. 9 <https://github.com/gomex/docker-para-desenvolvedores/blob/master/manuscript/organizacao.md> [↑](#footnote-ref-12)
13. 10 https://12factor.net/pt\_br/ [↑](#footnote-ref-13)
14. 6 https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [↑](#footnote-ref-14)