Profiscientia
Periódico Multidisciplinar do IFMT – Campus Cuiabá

# Alta Disponibilidade em *containers* Docker por meio do Docker Swarm

# High Availability in Docker containers by Docker Swarm

Luiz Carlos da Silva Filho[1]

Roberto Benedito de Oliveira Pereira[2]

**Resumo**

A virtualização é uma tecnologia que proporciona a execução simultânea de dois ou mais sistemas operacionais ou aplicações em uma máquina física. Com a virtualização é possível otimizar o uso dos recursos de uma máquina física por meio da distribuição desses recursos entre vários usuários ou ambientes. Além de proporcionar a otimização de uso, a virtualização deve prover ou ser amparada por técnicas ou mecanismos que garantam a sua execução durante o maior tempo possível e que seja tolerante à falhas. Este trabalho aborda a ferramenta Docker, que é baseada na virtualização por *containers*, e o Docker Swarm, responsável por prover a alta disponibilidade de *containers*, permitindo que serviços computacionais estejam disponíveis o maior tempo possível.

**Palavras-chave:** virtualização, *containers*, Docker, alta disponibilidade

[1] Bacharel em Ciência da Computação pela Universidade Federal de Mato Grosso. Email: lcdsf1@hotmail.com

[2] Professor Adjunto da Universidade Federal de Mato Grosso. Email: roberto@ic.ufmt.br

**Abstract**

Virtualization is a technology that can execute more than one operating system or application simultaneously in a single physical machine. With virtualization is to possible optimize the usage of resources from a physical machine by distributing its resources among users or environments. Besides providing usage optimization, virtualization must also provide or be aided by techniques or mechanisms that guarantee its uptime for as long as possible, with fault tolerance capacity. This paper approaches the Docker virtualization tool, that is based on container virtualization, and Docker Swarm, a native Docker tool that provides container high availability, thus, making critical computational service available as long as possible.

**Keywords:** virtualization, containers, Docker, high availability

## 1 INTRODUÇÃO

A virtualização é uma tecnologia que proporciona a execução simultânea de dois ou mais sistemas operacionais ou aplicações em uma máquina física (MENEZES; MATTOS, 2008). Com a virtualização é possível utilizar todos os recursos de uma máquina física por meio da distribuição desses recursos entre vários usuários ou ambientes (REDHAT, 2017).

Nos últimos anos, a aplicação da técnica de virtualização de servidores tem tomado uma representatividade considerável no mercado de trabalho, pois é um método que pode executar vários sistemas operacionais e aplicações em uma única máquina (BUI, 2015). Vários benefícios são obtidos com a virtualização, como a otimização do uso dos recursos computacionais, a redução de custos, a abstração dos recursos computacionais (LI, 2015) e a flexibilidade no desenvolvimento de sistemas devido à sua capacidade de alocação de recursos de hardware e software de forma dinâmica (NAIK, 2016).

A virtualização, atualmente, é baseada em dois métodos: máquinas virtuais (também conhecido como *hypervisor*) ou *containers*, que serão detalhados no decorrer do trabalho. A virtualização por máquina virtual — também conhecida como VM — é o método utilizado pela maioria das empresas (BERNSTEIN, 2014), porém, a virtualização por *containers* tem crescido consideravelmente nos últimos anos, pois é um método que exige menos recursos que as máquinas virtuais, sendo possível implantar e migrar serviços em um tempo menor. (NAIK, 2016). A ferramenta de virtualização por *containers* denominada Docker é consideravelmente utilizada atualmente, sendo a líder mundial deste segmento no mercado (DOCKER. 2017).

Logo, o objetivo geral do trabalho é prover alta disponibilidade em sistemas de *containers* Docker por meio do Docker Swarm e observar como o sistema se comporta em situações de falhas rotineiras,como a queda de um serviço do servidor ou até mesmo um erro de sistema.

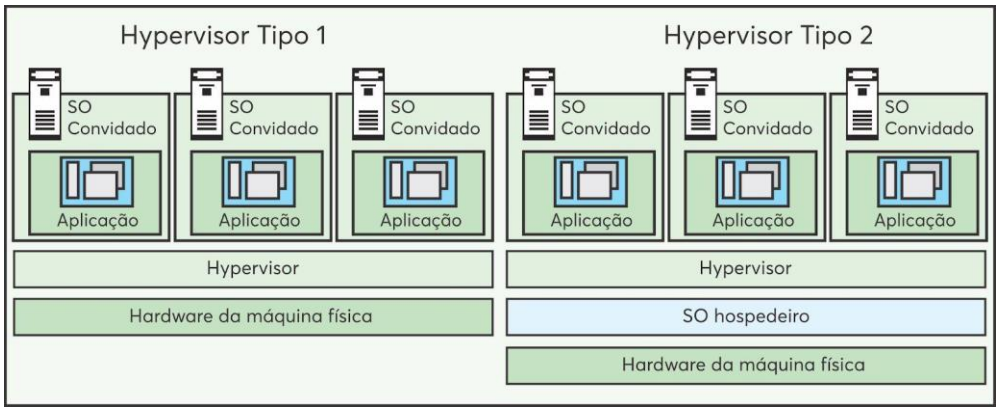
## 2 FUNDAMENTAÇÃO TEÓRICA

### 2.1 Virtualização por *Hypervisor* (máquinas virtuais)

A virtualização neste método ocorre em nível de hardware. Um *hypervisor* é o software responsável pelo gerenciamento de máquinas virtuais que são executadas sobre o Sistema Operacional (SO) hospedeiro, ou seja, da própria máquina física. Cada máquina virtual possui o seu próprio SO (conhecido como convidado) com o seu próprio núcleo, aplicações e suas dependências (binários e bibliotecas).

Existem dois tipos de *hypervisors* (Figura 1): Tipo 1 e Tipo 2. O Tipo 1, também conhecido como *hypervisor* bare metal, funciona diretamente sobre o hardware da máquina física e o Tipo 2, que funciona sobre o SO hospedeiro. A camada a mais presente no Tipo 2 faz com que o Tipo 1 se sobressaia no quesito performance (BUI, 2015). As principais ferramentas de virtualização que trabalham com *hypervisors* são: VMware, Xen, VirtualBox, KVM e Hyper-V.

Figura 1. Arquiteturas dos dois tipos de *hypervisors*




**Fonte:** (Adaptado de THOLETI, 2011)

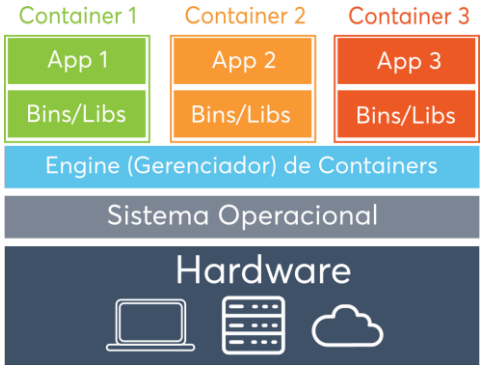
## 2.2 Virtualização por *Container*

A virtualização por *container*, diferente dos hypervisors, ocorre em nível de sistema operacional, ou seja, os *containers* estão sempre sobre o sistema operacional hospedeiro. É um tipo de virtualização que utiliza o núcleo do sistema operacional hospedeiro para executar múltiplos ambientes (BUI, 2015).

Os *containers* não exigem sistemas operacionais convidados, isso faz com que este método de virtualização exija menos recursos da máquina física (enquanto uma máquina virtual possui um tamanho em nível de gigabytes, um *container* está em nível de megabytes), entretanto, é possível executar sistemas operacionais em *containers*. No ponto de vista do sistema operacional hospedeiro, os *containers* são executados como

processos. Os recursos que eles utilizam podem ser compartilhados com a máquina física ou alocados separadamente para cada *container* (BUI, 2015). Nos *containers* são executados apenas as aplicações e suas dependências (binários e bibliotecas). O esquema básico da virtualização por *container* é apresentado na Figura 2 (SHAVERS, 2017). As principais ferramentas de virtualização baseada em *containers* são LXC (Linux Containers), Azure Container Service e Docker.

Figura 2. Arquitetura básica da virtualização por *containers*




**Fonte:** (Adaptado de SHAVERS, 2017)

## 2.3 Alta Disponibilidade

Atualmente existe uma dependência considerável por serviços computacionais para realizar tarefas de diversas naturezas, inclusive tarefas críticas, cujas falhas causariam prejuízos materiais, financeiros ou até mesmo perda de vidas. No intuito de mitigar esses prejuízos, são utilizadas diversas técnicas que visam garantir a disponibilidade desses serviços (FILHO, 2004).

A demanda crescente por infraestruturas computacionais designadas a servirem sistemas críticos faz com que a HA se torne um recurso de considerável importância atualmente (Heidi, 2016). Um sistema de HA tem o objetivo de estar o maior tempo possível disponível para que o funcionamento de seus serviços não sejam interrompidos, pois, todo serviço ou sistema está suscetível à falhas (PEREIRA, 2005).

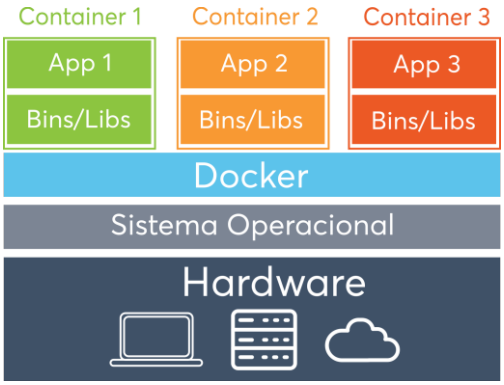
As falhas comuns em uma infraestrutura computacional podem ser de natureza física, causada pelo mau funcionamento de algum componente e que podem ser originadas por eventos como curtos-circuito ou o fim da vida útil de um ou mais componentes. De natureza humana, por meio da má elaboração no projeto de sistemas,

assim como erros no processo de manutenção e operação de sistemas (FILHO, 2004). É possível aplicar técnicas para contornar essas falhas, que são baseadas na replicação de hardware e software (BASELINE DATA, 2016).

## 2.4 Docker

O Docker é a ferramenta de virtualização por *containers*. Os desenvolvedores de software a utilizam para eliminar problemas de compatibilidade em seus programas. Operadores utilizam o Docker para executar e gerenciar as suas aplicações lado a lado em *containers* isolados para obter uma melhor densidade computacional. Empresas usam o Docker para acelerar o desenvolvimento e a integração dos seus softwares (Docker, 2016). A ferramenta é instalada sobre o sistema operacional hospedeiro, conforme a Figura 3.

Figura 2. Arquitetura do Docker




**Fonte:** (Adaptado de DOCKER, 2016)

## 2.5 Docker Swarm

Disponibilizada a partir da versão 1.12 do Docker, o Docker Swarm é uma ferramenta nativa do Docker para a orquestração de clusters Docker. No contexto do Docker Swarm, um swarm é um cluster de nós nos quais é possível implantar serviços. Um nó é uma instância do Docker em execução. É possível executar um ou mais nós em uma máquina (física ou virtual), mas o mais comum é a distribuição dos nós em múltiplas máquinas (DOCKER DOCUMENTATION, 2017C).

O Docker Swarm dispõe de um gerenciamento de cluster integrado ao Docker,

que permite utilizar o console do Docker para criar swarms nos quais são implantados aplicações (DOCKER DOCUMENTATION, 2017C).

No Docker Swarm, é possível declarar o número de tarefas a serem executadas para cada serviço desejado pelo usuário. Quando se escala para mais ou para menos, o gerenciador de swarm adapta automaticamente por meio da adição ou remoção de tarefas para manter o estado desejado. O Docker Swarm dispõe da funcionalidade de balanceamento de carga, na qual é possível expor as portas de serviços para um balanceador de carga externo. Internamente, o gerenciador de swarm permite que o usuário especifique como distribuir os *containers* de serviço entre os nós (DOCKER DOCUMENTATION, 2017C).

Quanto à HA, o Docker Swarm utiliza o gerenciador de swarm, que monitora constantemente o estado do swarm e reconcilia qualquer diferença entre o estado atual e o estado desejado pelo usuário. Por exemplo, um usuário define um serviço para ser executado em 10 réplicas de um *container*, e uma máquina que hospeda 2 dessas réplicas falha, o gerenciador de swarm atribui novas réplicas às outras máquinas que estão disponíveis e em execução (DOCKER DOCUMENTATION, 2017C).

## 3 MATERIAIS E MÉTODOS

### 3.1 Cenários de teste

Cada máquina virtual possui 1 unidade de armazenamento com RAID 5 composto por 3 SSDs Samsung Evo 850 com capacidade de 50GB, 8GB de memória RAM, 1 CPU Intel Xeon E52660 de 2.6GHz com 4 núcleos. O sistema operacional utilizado nas máquinas virtuais é o Ubuntu Server.

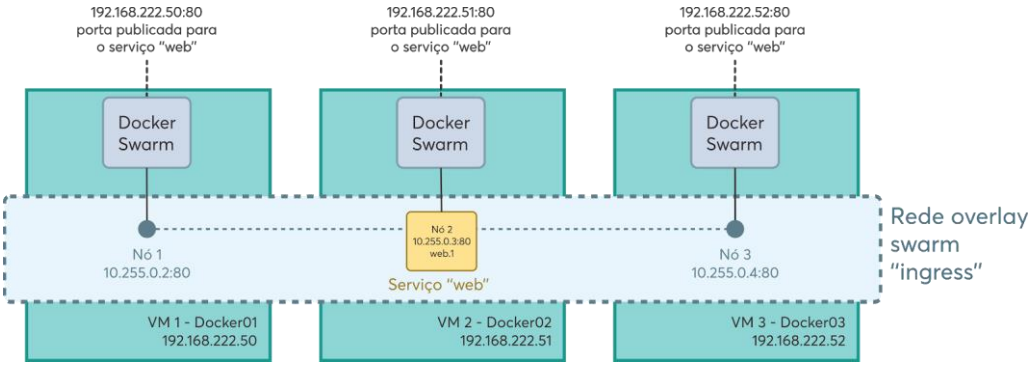
Foram estabelecidos dois cenários, suas estruturas estão representada na Figura 4 e 5. As três máquinas virtuais executarão o Docker Swarm e em ambos os ambientes executarão apenas um container do serviço, no intuito de confirmar a capacidade de balanceamento de carga e HA do swarm.

O swarm se encontra em uma rede sobreposta à rede das VMs, esta rede possui por padrão o nome "ingress" e é criada assim que um serviço é criado no swarm. Há a possibilidade da criação de uma rede personalizada, por meio do comando *docker network create*, no qual é possível definir o nome da rede, tipo do IP, gateway, entre

outras configurações.

Na figura 4, tem-se o ambiente destinado ao Estudo de Caso 1, neste cenário será utilizada uma Imagem Docker de um servidor web Nginx, o nome do serviço é "web", portanto, este serviço se trata de um servidor de páginas da Internet. A Imagem do Nginx foi utilizada por ser oficial e a mais popular do Docker Hub (DOCKER HUB, 2018), além do fato de o Nginx ser um dos servidores web mais utilizados no mercado atualmente, estando em terceiro lugar e com uma adoção crescente (NETCRAFT, 2018).

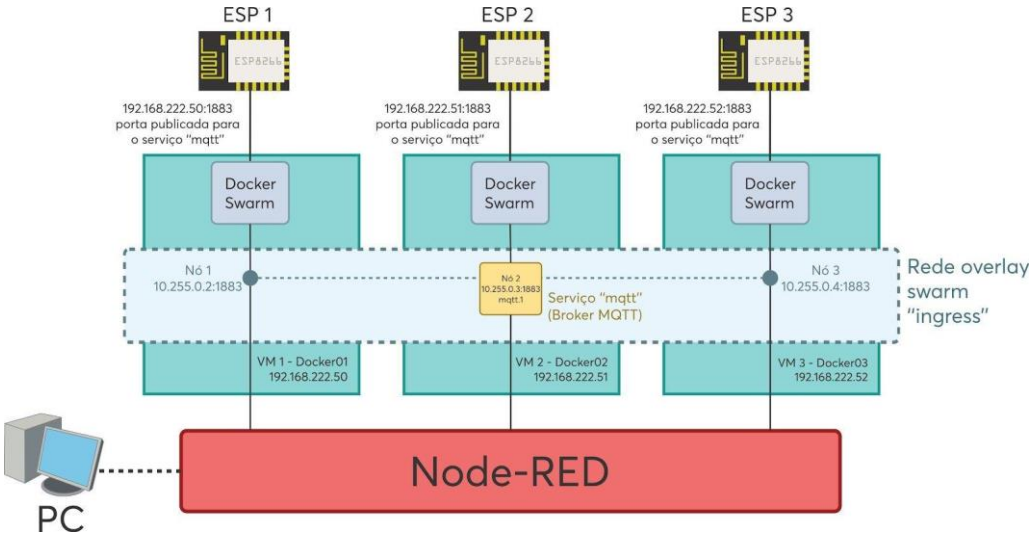
Figura 4. Esquema do cenário do serviço web




Para os estudos de caso 2 e 3, tem-se um ambiente que realiza a integração de sistemas distintos, pois executa um serviço de Broker MQTT Mosquitto, de nome "mqtt", representado na Figura 5. O Mosquitto é um servidor de mensagens para dispositivos IoT que possui a característica de ser leve, sendo possível executá-lo até mesmo em microcontroladores (ECLIPSE FOUNDATION, 2018).

Além das VMs e do Docker Swarm, o ambiente é composto por três microcontroladores ESP8266, que serão responsáveis pelo envio de mensagens para o Broker MQTT. O Node-RED, que se trata de uma ferramenta utilizada para a depuração da programação de microcontroladores, que neste caso foi utilizado para a visualização das conexões entre os ESPs e o Broker, assim como os dados enviados. O PC é o meio que permite de fato visualizar o progresso dos testes, por meio da sua conexão ao Node-RED.

Figura 5. Esquema do cenário do serviço mqtt




## 3.2 Estudos de caso

Como citado na seção anterior, foram definidos estudos de caso para a avaliação da HA do Docker Swarm, os sistemas a serem executados nos serviços são o servidor web Nginx e o Broker MQTT Mosquitto. A seguir, têm-se os detalhes de cada estudo de caso a ser estudado com a ferramenta Docker Swarm.

### 3.2.1 Estudo de caso 1: Balanceamento de carga

Será observado como o Docker Swarm realiza a o balanceamento de carga ao solicitar um serviço à uma VM que não esteja executando o container deste serviço, que será criado por meio do comando *docker service create --publish 80:80 --name web nginx*, sendo que: *--publish 80:80* expõe a porta 80 dos containers para a porta 80 da VM (é possível utilizar portas diferentes, neste caso foi utilizada a 80 para o *container* e para as VMs pelo fato desta porta ser a padrão do protocolo HTTP), *--name web* define que o nome do serviço será "web", e por fim, *nginx* é a Imagem Docker que será utilizada no serviço. Logo após a execução do comando, o Docker Swarm informará o andamento da criação do serviço e o seu status final (sucesso ou falha), será possível acompanhar o status do serviço após a sua criação, por meio do comando *docker service*

*ps web*. O container será executado em apenas uma das três VMs.

### 3.2.2 Estudo de caso 2: Situação de falha de VM

Será uma continuação do Estudo de Caso 1, pois foi utilizado o mesmo ambiente, porém, a VM que executava o único container será desligada. Durante a execução do Docker Swarm, uma das máquinas virtuais será desligada, simulando uma falha. Será observado como Docker Swarm contornará este problema para garantir a continuidade da execução dos sistemas conforme o estado desejado.

### 3.2.3 Estudo de Caso 3: Alta disponibilidade no Broker MQTT com balanceamento de carga

Será observado como o swarm (cluster de containers) do Docker Swarm se comporta em um sistema integrado. Este problema consiste na execução de um serviço de Broker MQTT no qual os microcontroladores ESP8266 estarão conectados um em cada VM por meio de um IP e uma porta, o serviço "mqtt" executará somente um container em uma VM, o objetivo é verificar se o swarm será capaz de realizar o balanceamento de carga, ou seja, redirecionar requisições do serviço das VMs que não possuem containers do serviço diretamente ao único container em execução.

## 4 RESULTADOS E DISCUSSÕES

### 4.1 Estudo de Caso 1

Figura 6. Comandos *docker service create* e *docker ps*

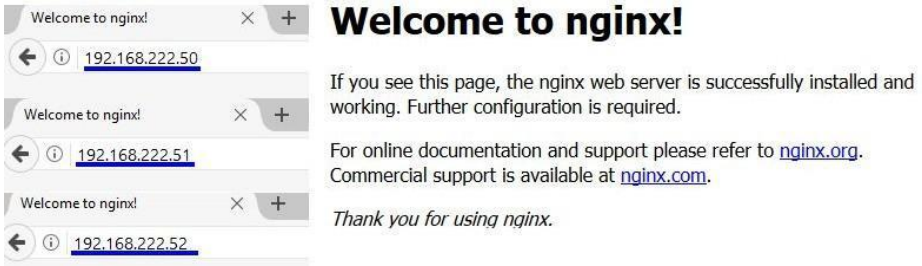
```
root@Docker01:/home/luiz# docker service create --publish 80:80 --name web nginx
u7ml512o077d6off07vm6nasq
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
root@Docker01:/home/luiz# docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE
ixoihact9aso        web.1               nginx:latest        Docker02            Running
root@Docker01:/home/luiz#
```

Na Figura 6 tem-se o resultado do comando da criação do serviço web, que foi

criada com sucesso. Em seguida, tem-se a saída do comando *docker ps web*, que apresenta o status atual do serviço "web", exibindo o ID, o nome e em qual nó (VM) o container está sendo executado.

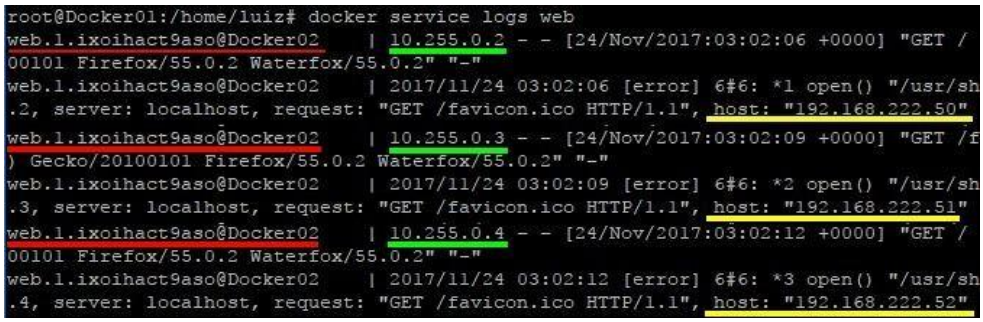
Foi realizado um teste de acesso à página padrão do Nginx por meio de um navegador, na Figura 7 é possível visualizar que, apesar de apenas uma VM executar o container do serviço, foi possível acessar pelos IPs de cada uma das 3 VMs (destacados pela linha azul), dessa forma comprovando que o Docker Swarm realiza o balanceamento de carga.

Figura 7. Acesso ao serviço "web" por meio das 3 VMs sendo realizado em um navegador




Outra prova do balanceamento de carga pode ser visto na Figura 8. O Nginx possui um recurso de log acessível por meio do comando *docker service logs web*, que exibe o container utilizado (destaque vermelho), os respectivos IPs da rede "ingress" de cada VM (destaque verde), assim como os seus IPs reais (destaque amarelo).

Figura 8. Comando *docker service logs web*




## 4.2 Estudo de Caso 2

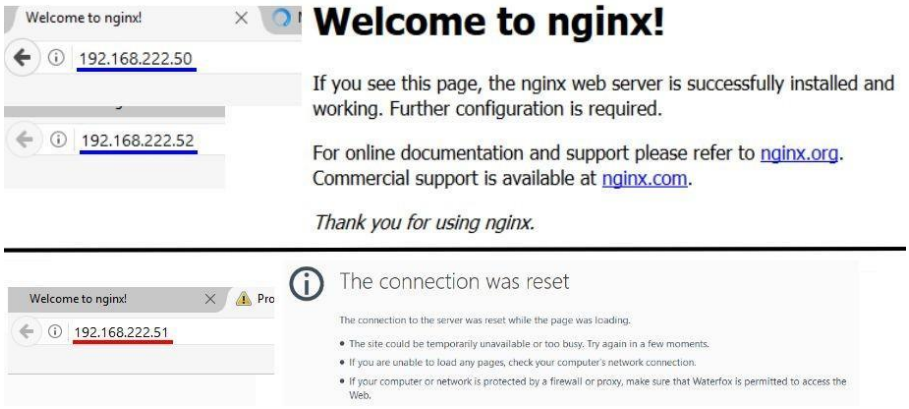
A VM desligada foi a Docker02. Na Figura 9, pode-se visualizar o que acontece logo após o seu desligamento. O comando docker node ls lista os nós e o status de cada

um, a Docker02 está desativada (down), o comando docker service ps web mostra que o container que anteriormente estava na Docker02 (destaque vermelho) foi migrado para a Docker03 (destaque azul), dessa forma, garantindo a continuidade do serviço.

Figura 9. comandos *docker node ls* e *docker service ps web*


A Figura 10 mostra o acesso ao serviço por meio do navegador, desta vez, a Docker02 estava indisponível (destaque vermelho), enquanto as outras VMs continuaram ativas (destaques azuis), com o Docker Swarm realizando o balanceamento de carga e garantindo a HA do serviço "web".

Figura 10. Acesso ao serviço web sendo realizado por meio de um navegador




Ao executar o comando docker service logs web (Figura 11) é possível verificar que a VM Docker02 não está disponível. Os acessos às outras VMs foi realizada com êxito, desta vez, com o container sendo executado na VM Docker03 (destaque azul). É possível conferir que o ID do nó que representa a VM Docker02 (Destaque amarelo) é o mesmo ID destacado na Figura 9.

Figura 11. Comando *docker node ls* e *docker service logs web* após o desligamento da VM Docker02

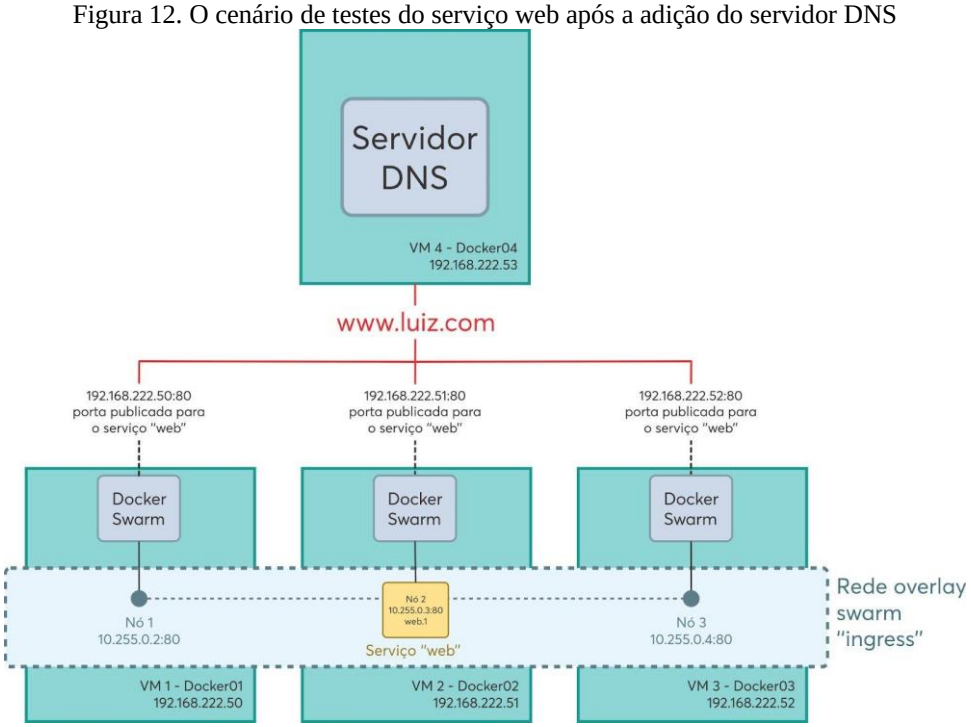
```
root@Docker01:/home/luiz# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGE
kdggl7oav8h37ekz8x2vs5jo7 *   Docker01            Ready               Active              Leader
ewwnlw466gmlzmc3p35ycaonn     Docker02            Down                Active              Unreac
vkcn8wgfdrv9siplm0887ryid     Docker03            Ready               Active              Reacha
root@Docker01:/home/luiz# docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE
lfurm3nvbg8t        web.1               nginx:latest        Docker03            Running
ixoihact9aso         \_ web.1           nginx:latest        Docker02            Shutdown
root@Docker01:/home/luiz#
```

### 4.2.1 O fator da limitação de acesso às máquinas e a solução por meio de um servidor DNS

Apesar do Docker Swarm realizar o balanceamento de carga, existe uma limitação, não é possível definir um endereço IP ou um nome de serviço que sirva para acessar as 3 VMs, portanto, não é possível acessar o serviço "web" por meio de um IP de uma máquina com falha (screenshot do meio na Figura 10).

Para contornar este problema, foi utilizada a associação de uma URL "www.luiz.com" aos 3 endereços IP das VMs, esta associação foi realizada por meio de um servidor DNS (mais detalhes no Anexo B), o que causou uma alteração no cenário de testes, como demonstrado na Figura 12.

Figura 12. O cenário de testes do serviço web após a adição do servidor DNS

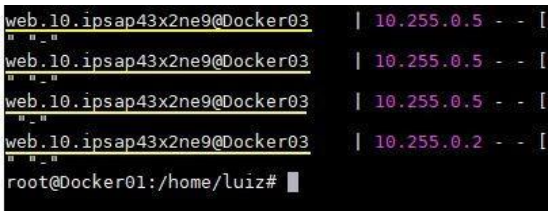



Para a validação desta solução, foram realizados testes com o serviço "web"

sendo executado com apenas 1 container e posteriormente com 10 containers. Em ambos os testes foram realizados 4 acessos simultâneos à URL definida.

O serviço "web" foi executado em um único container. O acesso simultâneo à URL foi feita por meio do envio de um comando às 4 VMs ao mesmo tempo. A realização de comandos simultâneos foi realizado por meio do cliente SSH MobaXterm. As 4 VMs conseguiram acessar a página padrão do nginx. Ao observar o relatório do serviço "web", nas últimas 4 linhas, tem-se o registro dos 4 últimos acesso à página do serviço por meio do único container em execução (Figura 13).

Figura 13. Relatório do serviço web, com os 4 últimos registros de acesso à página do serviço




A Figura 14 apresenta a lista de containers do serviço "web" para o segundo teste, sendo 10 réplicas. Aqui o procedimento foi o mesmo do teste anterior.

Figura 14. O serviço web executando 10 containers

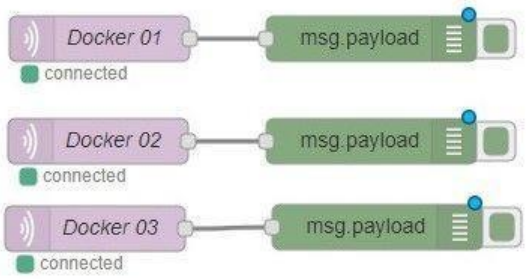

Desta vez, o relatório do serviço "web" (Figura 15) mostra que 3 dos 10 containers atenderam a requisição, assim provando que o uso de um servidor DNS pode ser um complemento no balanceamento de carga do Docker Swarm.

Figura 15. Relatório do serviço web sendo executado em 10 containers, com os 4 últimos registros de acesso à página do serviço


## 4.3 Estudo de Caso 3

O serviço "mqtt" executou um Broker MQTT que recebeu as mensagens dos ESPs, basicamente, o serviço consiste no envio de mensagens do ESP8266 para o Broker MQTT. Para a visualização dessas mensagens, assim como o acompanhamento geral da conexão entre os ESPs e o Broker MQTT, foi utilizada a ferramenta Node-RED. Na Figura 16 é possível visualizar as 3 conexões deste teste, todas online.

Figura 16. Representação das três conexões entre os ESP8266 e as VMs




É possível verificar a conexão dos três ESPs no único container do serviço "mqtt" por meio da visualização do log do serviço. Para isso foi necessário acessar o container e executar o comando tail -f mosquitto.log. Na Figura 17 tem-se a saída do comando, mostrando que o container recebeu conexões das três VMs, por meio dos IPs da rede "ingress" e a porta que foi exposta no serviço "mqtt" (destaque verde).

Figura 17. Execução do comando *tail -f mosquitto.log* dentro do container

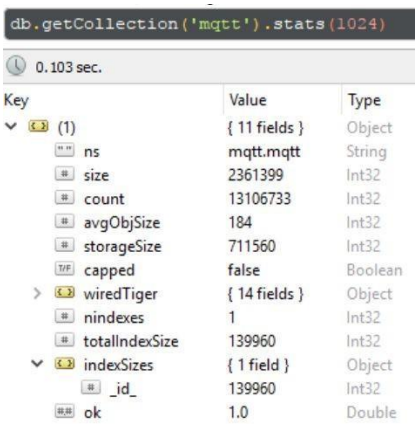

O Node-RED permite a visualização das mensagens que são enviadas pelos

ESPs ao Broker. No destaque é possível visualizar 3 endereços MAC distintos (Figura 18, destaques azuis), cada um está associado a um ESP8266. Os dados enviados pelos ESPs são referentes à temperatura, umidade e luminosidade de uma sala.

Figura 18. Visualização dos dados enviados pelos ESPs ao Broker MQTT


É importante ressaltar que o serviço "mqtt" foi utilizado como Broker MQTT de um Trabalho de Conclusão de Curso sobre persistência de dados do protocolo MQTT em um cluster de banco de dados MongoDB, que também foi executado como um serviço no Docker Swarm (MIRANDA, 2017). A captura de tela na Figura 19 mostra alguns detalhes acerca dos dados trafegados pelo Broker do serviço "mqtt", nota-se que passaram 13.106.733 dados pelo serviço.

Figura 19. Detalhes dos dados trafegados pelo serviço mqtt (MIRANDA, 2017)



## 5 CONCLUSÕES

Pelos resultados obtidos, foi constatado que o Docker Swarm é uma ferramenta que consegue alcançar a HA por meio da migração de containers de um nó para o outro, bem como a capacidade de realizar o balanceamento de carga dos seus serviços por meio do redirecionamento de conexões de nós que não possuem container para aqueles que possuem, embora necessite de ferramentas externas para redirecionar as solicitações de acesso aos seus serviços.

No Estudo de Caso 1 foi possível verificar a capacidade de balanceamento de carga do Docker Swarm por meio de um serviço de um servidor web, sendo possível realizar o acesso ao serviço por meio dos 3 IPs, apesar de apenas uma das VMs executar o container deste serviço.

No Estudo de Caso 2, foi verificada a capacidade de HA no serviço "web", por meio de uma simulação de falha de VM. O Docker Swarm foi capaz de realizar a migração do container da máquina com falha para outra que estava disponível no swarm, entretanto, foi preciso utilizar um servidor DNS para realizar o balanceamento de carga ao acessar o serviço por meio de uma única URL.

O Estudo de Caso 3 mostra que o Docker é capaz de trabalhar com integração de sistemas distintos, sendo possível realizar a conexão de três microcontroladores aos nós do swarm, bem como a capacidade de balanceamento de carga em ambientes desta natureza, ou seja, com sistemas distintos.

Conclui-se que o Docker Swarm é uma ferramenta ideal para a execução de serviços críticos devido ao seu balanceamento de carga e a garantia de HA em seus serviços.

## 6 REFERÊNCIAS

BERNSTEIN, D. Containers and cloud: From LXC to docker to kubernetes. IEEE Cloud Computing, 2014.

BUI, T. Analysis of Docker Security. 2015.

DATA, B. System High Availability and Hardware High Availability – What’s the

Difference? Baseline Data Services, 2016. Acesso em 13 de agosto de 2017. Disponível em: <https://baseline-data.com/blog/high-availability/system-hardware-high-availability-differences/>.

DOCKER. Docker for the Virtualization Admin. 2016. Disponível em: <https://goto.docker.com/rs/929-FJL-178/images/Docker-for-Virtualization-Admin-eBook.pdf>.

DOCUMENTATION, D. About images, containers, and storage drivers. Docker Inc, 2017. Acesso em 11 de agosto de 2017. Disponível em: <https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#images-and-layers>.

DOCUMENTATION, D. Docker Glossary. Docker Inc, 2017. Acesso em 11 de agosto de 2017. Disponível em: <https://docs.docker.com/glossary/?term=image>.

DOCUMENTATION, D. Overview of Docker Hub. Docker Inc, 2017. Acesso em 11 de agosto de 2017. Disponível em: <https://docs.docker.com/docker-hub/>.

DOCUMENTATION, D. Swarm mode overview. Docker Inc, 2017. Acesso em 14 de agosto de 2017. Disponível em: <https://docs.docker.com/engine/swarm/#feature-highlights>.

FILHO, N. A. P. Serviços de Pertinência para Clusters de Alta Disponibilidade. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, Brasil, 2004. 23 FOUNDATION, ECLIPSE. Eclipse Mosquitto™ - An open source MQTT broker. Acesso em 06 de novembro de 2018. Disponível em <https://mosquitto.org/>.

HEIDI, E. What is High Availability? DigitalOcean, 2016. Acesso em 20 de setembro de 2017. Disponível em: <https://www.digitalocean.com/community/tutorials/what-is-high-availability>.

HUB, DOCKER. Explore Official Repositories. Acesso em 06 de novembro de 2018. Disponível em: <https://hub.docker.com/explore/>.

INC, D. Hardware and software requirements. Docker Inc, 2017. Acesso em 11 de agosto de 2017. Disponível em: <https://docs.docker.com/datacenter/ucp/1.1/installation/system-requirements/>.

INC, D. What is Docker? Docker Inc, 2017. Acesso em 9 de julho de 2017. Disponível em: <https://www.docker.com/what-docker>.

LI, W.; KANSO, A. Comparing containers versus virtual machines for achieving high availability. In: Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015. [S.l.: s.n.], 2015]

MENEZES, D.; MATTOS, F. Virtualização: VMWare e Xen. 2008.

MERKEL, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, v. 2014, n. 239, 2014.

MIRANDA, T. L. R.; PEREIRA, R. B. d. O. Implementação de um Cluster de banco de dados no Raspberry Pi com MongoDB para replicação e persistência dos dados IoT. 2017.

NAIK, N. Building a virtual system of systems using docker swarm in multiple clouds. In: ISSE 2016 - 2016 International Symposium on Systems Engineering - Proceedings Papers. [S.l.: s.n.], 2016.

NETCRAFT. August 2018 Web Server Survey. Acesso em 06 de novembro de 2018. Disponível em <https://news.netcraft.com/archives/2018/08/24/august-2018-web-server-survey.html>

PEREIRA, R. B. O. Alta Disponibilidade em Sistemas GNU/LINUX utilizando as ferramentas Drbd, Heartbeat e Mon. 2005. 23

REDHAT. What is virtualization? RedHat, 2017. Acesso em 7 de julho de 2017. Disponível em: <https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization>.

ROUSE, M. What is legacy application? TechTarget, 2017. Acesso em 01 de agosto de 2017. Disponível em: <http://searchitoperations.techtarget.com/denition/legacy-application>.

SHAVERS, M. 3 Drastic Reasons Containers are Causing a Seismic Shift in Technology. LinkedIn, 2017. Acesso em 9 de agosto de 2017. Disponível em: <https://www.linkedin.com/pulse/3-drastic-reasons-containers-causing-seismic-shift-mark-shavers>.

THOLETI, B. P. Learn about hypervisors, system virtualization, and how it works in a cloud environment. IBM, 2011. Acesso em 15 de agosto de 2017. Disponível em: <https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/index.htm>