Uso de contêineres para desenvolvimento de infraestrutura multi-nuvem para aplicações de *Smart Farming*

Diogo José da Silva Ribeiro[1], Júnio César de Lima[2], Júlio César Ferreira[3], Gabriel da Silva Vieira[4]

[1]Instituto Federal Goiano – Campus Urutaí

[2]Núcleo de Informática – Instituto Federal Goiano – Campus Urutaí.

[1]diogo.ribeiro@estudante.ifgoiano.edu.br

{[2]junio.lima, [3]julio.ferreira, [4]gabriel.vieira}@ifgoiano.edu.br

Abstract. *Internet of Things refers to the digital interconnection of everyday services with the Internet, forming a network of physical objects capable of collecting and transmitting data. Internet of Things-based systems are used in several application domains, including e-commerce, Smart Cities and Smart Farming. Currently, construction systems for the Internet of Things have several research challenges, of which the implementation of applications, using the concept of containers on computational resources available in the cloud, is an important line. This article proposes and presents a multi-cloud infrastructure based on containers for the implementation of applications focused on Smart Farming. Initial results show that the use of clusters, independent of the cloud provider, increases the scalability and availability of the deployed applications.*

Resumo. *A Internet das Coisas se refere à interconexão digital de serviços do cotidiano com a Internet, formando uma rede de objetos físicos capazes de coletar e transmitir dados. Os sistemas baseados em Internet das Coisas são usados em vários domínios de aplicação, incluindo e-commerce, Smart Cities e Smart Farming. Atualmente a construção de sistemas para a Internet das Coisas possuem vários desafios de pesquisa, dos quais a implantação de aplicações, utilizando o conceito de contêineres sobre recursos computacionais disponibilizados na nuvem, é uma linha importante. Este artigo propõe e apresenta uma infraestrutura multi-nuvem baseada em contêineres para implantação de aplicações voltadas para a Smart Farming. Resultados iniciais mostram que a utilização de clusters, independentes do provedor de nuvem, aumenta a escalabilidade e disponibilidade das aplicações implantadas.*

1. Introdução

Com a consolidação da era da Internet das Coisas (*Internet of Things* - IoT) há a necessidade de uma quantidade cada vez maior de dispositivos conectados, contribuindo com o crescimento exponencial de tráfego, armazenamento e gerenciamento de quantidades cada vez maiores de dados. Além disso, o aumento da demanda de recursos computacionais implica na necessidade de otimização destes recursos. Neste contexto, a computação em nuvem é uma grande aliada, se tornando a responsável por prover toda a infraestrutura necessária para aplicar os conceitos de Internet das Coisas.

O desenvolvimento de sistemas na era da IoT deve ser capaz de abordar a complexidade, a heterogeneidade, a interdependência e, especialmente, a evolução de sistemas em rede fracamente conectados [Bertolino et al. 2011]. O poder de compor serviços independentes em serviços com maior granularidade promove ainda mais produtividade e reutilização, onde os serviços podem ser utilizados para agasalhar aplicações existentes, bem como para desenvolver novas aplicações.

Do ponto de vista das aplicações para IoT, em especial dentro do contexto de *Smart Cities* e *Smart Farming*, a composição de serviços pode reduzir o custo e os riscos da construção de novas aplicações, uma vez que as lógicas de negócios existentes são representadas como serviços e podem ser reutilizadas.

Smart Farming refere-se a aplicação das tecnologias de informação na produção agrícola visando uma melhora na utilização dos recursos e assim, aumentar a produção. A maioria das aplicações no contexto de *Smart Farming* fazem uso de informações captadas por usuários, sensores, satélites e veículos aéreos remotamente controlados, onde tais conjuntos de dados necessitam ser previamente processados e, em seguida, integrados para gerar a solução requisitada pelos usuários [Bhange and Hingoliwala 2015].

O objetivo deste artigo é analisar e propor uma infraestrutura multi-nuvem baseada em contêineres para implantação de aplicações voltadas para a *Smart Farming*.Para este trabalho foi utilizado uma aplicação *web* para gerenciar botijões criogênicos de sêmen bovino que são utilizados no processo de inseminação artificial, esta aplicação tem como principal objetivo de gerenciar o nível de nitrogênio do botijão e otimizar o processo de busca das doses de sêmen dentro destes botijões.

A construção e disponibilização de aplicações utilizando a tecnologia de nuvem e contêineres contribuem com a melhoria na gestão eficiente de recursos computacionais e com a qualidade dos serviços, sendo um ponto importante em aplicações

A abordagem proposta consiste na utilização de *clusters*, independentes do provedor de nuvem. Para demostrar essa independência foi utilizado um *cluster* multi-nuvem, utilizando AWS e *Microsoft Azure*. Já para o *storage* foi utilizado *Amazon Elastic File System* e *Azure File Share*, *DockerEngine 19.03.11* e *DockerMachine 0.16.2*.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta o referencial teórico. A Seção 3 fornece a metodologia utilizada na implementação e implantação de infraestrutura proposta. A Seção 4 fornece os resultados dos testes realizados na infraestrutura proposta. Por fim, a Seção 5 são feitas as considerações finais sobre o trabalho realizado.

2. Referencial Teórico

Nos últimos anos a agropecuária está tendo que lidar de vez com as mudanças trazidas pela revolução digital, sendo este processo chamado de Smart Farming ou Agricultura Inteligente [Virk et al. 2020]. Smart Farming é baseada na integração das aplicações digitais e para isso precisa mesclar dispositivos móveis, sensores, atuadores, GPS, drones, imagens de satélites, robótica, Big Data e Internet das Coisas [Duft 2018].

A IoT é uma extensão da Internet, tipicamente com menor porte. Ela proporciona aos objetos do dia-a-dia dotados de capacidade computacional e de comunicação se conectarem à Internet. A conexão com a rede mundial de computadores viabiliza, primeiro,

o controle remoto dos objetos e, segundo, a utilização dos próprios objetos como provedores de serviços [Santos et al. 2016]. Logo, há um aumento da demanda de recursos computacionais, fazendo-se, então, necessário a otimização do uso destes recursos. Nesse sentido, a computação em nuvem tem a função de prover a infraestrutura de grandes empresas que utilizam os conceitos de Internet das Coisas.

Computação em nuvem (*Cloud Computing*) é um modelo que permite acesso a recursos computacionais sob medida e sob demanda, de maneira similar a recursos como água e eletricidade. Este paradigma possibilita acesso, através de rede, a um conjunto de recursos computacionais (rede, armazenamento, processamento, plataformas de desenvolvimento e aplicações), que podem ser rapidamente aprovisionados e liberados com um mínimo esforço de gerenciamento e interação por parte do provedor [Gomes 2017].

Dentre as tecnologias utilizadas na computação em nuvem, pode-se ressaltar a tecnologia de virtualização: a virtualização tradicional, baseada em máquinas virtuais, e conteinerização, baseada em contêineres. Na virtualização tradicional o uso da abordagem de máquinas virtuais permite a execução de sistemas em diferentes ambientes computacionais com garantia da manutenção das dependências originais. Nesse sentido, máquinas virtuais consistem em programas emuladores que simulam todas as funcionalidades de um computador em outro, possibilitando a substituição completa de uma máquina real [da Cruz et al. 2018].

Uma alternativa à virtualização tradicional é a virtualização baseada em contêineres. Um contêiner é o agrupamento de uma aplicação junto com as suas dependências que compartilham o *kernel* do sistema operacional do *host*, podendo ser tanto uma máquina virtual quanto física [Docker 2020]. Em um ambiente em nuvem, a utilização da virtualização baseada em contêineres permite uma otimização dos recursos disponíveis, uma vez que um contêiner é mais leve e mais integrado ao sistema operacional da máquina *host* que uma máquina virtual.

Atualmente existem inúmeras plataformas de contêineres, tais como: *Open Container Initiative* (OCI) [Initiative 2020], *Apache Mesos and Mesosphere* [Apache 2020] e *Docker* [Docker 2020]. Dentre essas plataformas, *Docker* tem sido amplamente utilizada em diferentes contextos IoT, em consonância com de computação em nuvem, em razão de suas características como leveza, velocidade e portabilidade [Trindade 2018].

O *Docker* é uma plataforma para desenvolvedores e administradores de sistemas que permite criar, executar e compartilhar aplicativos com contêineres [Docker 2020]. Essa plataforma, que é um projeto *open source*, apresenta um mecanismo de contêiner que fornece uma solução completa para criação e distribuição de contêineres. Além disso, o *Docker* possui ferramentas adicionais que permitem o gerenciamento de *clusters* e o provisionamento de novos *hosts Docker* [Mouat 2015].

No *Docker*, um contêiner é criado utilizando uma imagem que inclui toda a infraestrutura necessária para executar uma aplicação: o código ou binário, tempos de execução, dependências e quaisquer outros objetos do sistema de arquivos necessários [Docker 2020].

Contêineres normalmente são utilizados em *clusters*, que é um conjunto de máquinas interconectadas, também chamadas de nós, trabalhando em conjunto. Para a criação, gerenciamento e manutenção de um *cluster* são utilizadas ferramentas denomi-

nadas orquestradores. *Docker Swarm* e o *Kubernetes* são os exemplos mais comuns de orquestradores.

O *Docker Swarm* é o orquestrador nativo do *Docker* que possibilita a criação de um *cluster* em ambiente local e em nuvem. Ao lidar com *cluster*, o *Docker Swarm* cria um grupo cooperativo de sistemas que podem fornecer redundância, caso um ou mais nós falhem, por exemplo. Além disso, ele atribui contêineres aos nós subjacentes e otimiza os recursos, agendando automaticamente as cargas de trabalho do contêiner para serem executadas no *host* mais apropriado com os recursos adequados, mantendo os níveis de desempenho necessários [Rouse 2016].

O *Kubernetes* é um orquestrador *Open Source*, desenvolvido pela Google[Google 2020], com o objetivo de automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêiner. Além de permitir agrupar *hosts* executados em contêineres Linux em *clusters*, o *Kubernetes* auxilia no gerenciamento eficiente de *clusters*, permitindo incluir *hosts* em nuvem pública, nuvem privada, ou nuvem híbrida [Redhat 2020].

3. Metodologia

Para construção da abordagem proposta nesse artigo foi realizada uma revisão abrangente do estado-da-arte sobre como os conceitos de contêineres podem ser utilizados para implantação de sistemas relacionados à IoT, em especial *Smart Farming*, utilizando recursos fornecidos por provedores de recursos em nuvens. Ao utilizar os contêineres busca-se a otimização do uso dos recursos computacionais dispostos nos provedores de nuvem de forma a economizar recursos, inclusive financeiros, e aumentar a disponibilidade das aplicações.

Sendo assim, a metodologia foi dividida em três fases. A seguir, é apresentada a programação detalhada de cada uma fases.

3.1. Fase 1: Testes e modelagem dos resultados da revisão bibliográfica

Nesta fase foram levantadas as principais tecnologias sobre contêineres que são usadas para implantação de sistemas computacionais. Além disso, foram levantados os principais provedores de nuvem, como *Amazon Web Service* (AWS) [Amazon 2020], *Microsoft Azure* (*Azure*) [Microsoft 2020] e *Google Cloud Platform* [Google 2020], sendo que os testes foram realizados nas duas primeiras plataformas, uma vez que elas oferecem acesso gratuito aos recursos necessários para o teste.

Para a realização de testes na AWS foram utilizadas 3 máquinas virtuais do tipo *t2.micro* com *1vCPU*, 1 *GigaByte* de memória RAM, sistema operacional Ubuntu Server 18.04 LTS, e *Docker* 18.09. Na *Azure*, foram utilizadas 3 máquinas *B1s Standard* com as mesmas configurações de máquinas que na AWS. Na montagens dos *clusters*, em ambos provedores de nuvem foram utilizadas as plataformas *Docker Swarm* e o *Kubernetes*, onde foram montados *clusters* com 3 máquinas para testes de escalabilidade e de desempenho.

3.2. Fase 2: Definição de uma abordagem

Na segunda fase foi definida uma abordagem para implantação de sistemas utilizando recursos computacionais disponibilizados na nuvem. A definição da abordagem proposta

foi baseada na comparação das métricas, apresentadas na fase anterior, com foco em facilidades para uso de contêineres no contexto de aplicação em IoT implantadas em nuvens computacionais.

A abordagem consiste em um *cluster Docker Swarm*, que pode ser implantada independente do provedor de nuvem, desde que o mesmo possua um sistema de *storage*. Para demostrar essa independência foi utilizado um *cluster* multi-nuvem, utilizando AWS e *Azure*. Já para o *storage* foi utilizado *Amazon Elastic File System* e *Azure File Share*, *Docker Engine 19.03.11* e *Docker Machine 0.16.2*.

3.3. Fase 3: Desenvolvimento de serviços

Na última fase foram desenvolvidas aplicações para serem implantadas usando a abordagem proposta. Para isso, foram definidos problemas dentro da *Smart Farming*.

Nesse contexto, foram desenvolvidas duas aplicações voltados a área de *Smart Farming*, cujo objetivo principal é gerir de forma eficiente os botijões criogênicos de sêmen utilizados na inseminação artificial. Foram desenvolvidos um *web service* e uma aplicação *Web*, com as seguintes características descritas a seguir:

- *Web Service*
  O *web service* foi implementado utilizando a linguagem JAVA na versão 8, as bibliotecas Jersey na versão 2.27 e *Firebase* com SDK na versão 6.10.0, o servidor *Web* JAVA *Tomcat* na versão 8.5.54. Esse *web service* possibilita a comunicação de vários tipos de aplicação ao banco de dados dos botijões criogênicos de sêmen, via internet por meio do protocolo *HyperText Transfer Protocol* (HTTP).

- Aplicação *Web*
  A aplicação foi implementada utilizando as linguagens de desenvolvimento para *web* HTML, CSS e *JavaScript*. Para o armazenamento e autenticação foi utilizado o *Firebase* com a SDK na versão 5.0.1. Essa aplicação *web* consiste no sistema que gerencia de forma eficiente o manejo de botijões criogênicos de sêmen.

Depois de desenvolvidos, os protótipos foram implantados na AWS e *Azure*, seguindo a abordagem proposta na Fase 2.

4. Resultados

Essa seção mostra os principais resultados deste trabalho. Inicialmente, a subseção 4.1 discute os resultados dos testes com as ferramentas *Docker* e *Kubernetes*. Em seguida, a subseção 4.2 apresenta a infraestrutura utilizada para a implantação dos serviços relatados na subseção 3.3.

4.1. Comparativo entre *Docker Swarm* e *Kubernetes*

Inicialmente foram realizados testes utilizando os provedores de nuvem da AWS e *Azure* para orquestração dos *clusters*. Estes testes consistiram na utilização de contêineres dentro de um *cluster* utilizando dois orquestradores: o *Docker Swarm* e o *Kubernetes*.

Em um *cluster* existem dois tipos de nós: *leader/master* e *worker*. Em cada *cluster* existe apenas um nó do tipo *leader/master*, que é o nó principal, responsável por realizar o gerenciamento do *cluster*. A adição de um novo nó a um *cluster* é realizada através

da execução de um *join* passando a chave de acesso. Caso já exista um outro nó *leader/master* e seja inserido outros nós do tipo *leader/master*, eles serão marcados como *reachable*, ou seja, em caso de falha no nó *leader/master* atual um dos nós marcados como *reachable* será eleito para ser o novo *leader/master*.

Um nó do tipo *worker* somente é utilizado para execução dos contêineres. Para manter um *cluster* estável é necessário que 50% dos nós mais 1 seja do tipo *leader/master*, pois em caso de falha no *leader/master* atual um novo é selecionado de forma aleatória para assumir o seu lugar.

O teste utilizando o *Docker Swarm* e o *Kubernetes* consistiu em executar um contêiner com servidor *web Nginx* [Nginx 2020], iniciando com apenas 1 contêiner e aumentando progressivamente de 5 em 5 contêineres (1, 5, 10, ...). Após cada incremento foi verificado se todos os contêineres foram iniciados de forma correta e, em seguida, foram executado requisições HTTP ao *cluster*. Estes procedimentos se repetiram até atingirmos o limite do *cluster*, ou seja, quando o mesmo não conseguiu completar a ação de adicionar contêineres.

Os testes foram realizados visando analisar a escalabilidade e desempenho. Foi observado que o *Docker Swarm* possibilitou escalar o *cluster* em até 100 contêineres. Ao incrementar o *cluster* acima de 100 contêineres, o *Docker Swarm* começa a apresentar lentidão para criar novos contêineres, chegando a falhar. Ao tentar escalar acima de 130 contêineres, o *cluster* apresenta falha em todos os contêineres de todos os nós, podendo tornar impossível enviar comandos via terminal para qualquer nó deste *cluster*.

Os procedimentos realizados com o *Docker Swarm* foram realizados com o *Kubernetes*. Com o *Kubernetes* foi possível executar até 150 contêineres. Ao aumentar acima de 150 contêineres, é possível perceber um tempo maior para provisionar os novos contêineres, sendo que alguns contêineres não foram iniciados. Ao atingir o número de 200 contêineres, o *cluster* apresenta falha em todos os contêineres de todos os nós, tornando impossível enviar comandos via terminal para qualquer nó deste *cluster*.

O *Kubernetes* mostrou ter poder de escalabilidade maior que o *Docker Swarm*. Porém, o processo de instalação é mais complexo, requerendo algumas configurações adicionais. Como o *Docker Swarm* é integrado ao *Docker*, tornam-se desnecessárias configurações adicionais para utilizá-lo.

4.2. Implantação

Com base nos requisitos necessários para a implantação dos serviços, definidos na Seção 3.3, foi planejada uma infraestrutura para implantá-los. A infraestrutura tem como propósito mostrar a independência de plataforma que contêineres *Docker* podem fornecer.

A infraestrutura proposta foi formada por um *cluster* multi-nuvem (AWS e *Azure*) com 6 nós e os sistemas de *storage* dos provedores de nuvem utilizados. O modelo de *cluster* multi-nuvem consiste em criar um *cluster* em duas ou mais nuvens, permitindo utilizar os serviços de cada provedor. O uso de *cluster* multi-nuvem juntamente com a redundância de 6 nós permite maior disponibilidade das aplicações implantadas.

As aplicações foram implantadas em uma infraestrutura de *cluster Docker Swarm* multi-nuvem composto por 6 máquinas (nós), sendo 3 provisionadas na AWS e 3 provisionadas na *Azure*. Em cada máquina foi instalada a *Docker Engine*, que é responsável

por executar os contêineres com os serviços. Essa divisão entre os dois provedores foi realizada para garantir o balanceamento.

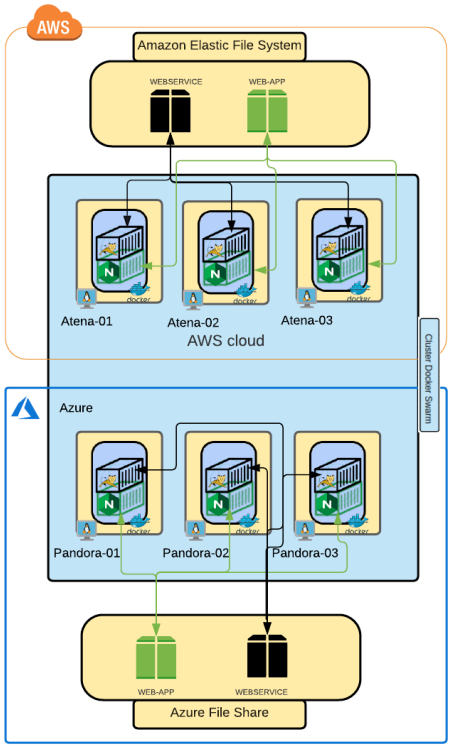

Figura 1. Infraestrutura de *cluster* multi-nuvem.

A infraestrutura utilizada para a implantação é mostrada na Figura 1, que ilustra os provedores de nuvem AWS e *Azure*. O provedor AWS contém 3 nós (Atena-01, Atena-02, Atena-03) e o sistema de *storage Amazon Elastic File System*. Já o provedor *Azure* contém também 3 nós (Pandora-01, Pandora-02, Pandora-03) e o sistema de compartilhamento de aquivos *Azure File Share*. Dentro de cada nó, o *Docker* executa 2 instâncias de contêineres: *Nginx* e *Tomcat*. Todos os nós estão conectados dentro de um *cluster Docker Swarm*.

Foram implantadas 6 instâncias de contêineres com *Nginx*, responsáveis pela execução da aplicação *web* e 6 instâncias de contêineres com *Tomcat*, responsáveis pelo *Web Service*. O sistema de balanceamento nativo do *Docker Swarm* busca distribuir os contêineres de forma igual entre os nós. Como foram definidas 6 instâncias de cada serviço, o *Docker Swarm* coloca uma instância de cada serviço. Dessa forma são executado 2 contêineres em cada nó.

Geralmente utiliza-se volumes locais para persistência dos dados em contêineres, mas quando se utiliza em *cluster* é recomendado que se utilize volumes compartilhados. Na infraestrutura proposta, foram utilizados como volumes compartilhados o *Amazon Elastic File System* e o *Azure File Share*, onde em ambos sistemas foram criados 2 volumes, sendo um para o *Web Service* e outro para a aplicação *web*. Os códigos fonte foram colocados dentro de seus respectivos volumes. Os contêineres consultam os códigos fonte nos volumes e os executam. Vale ressaltar que apesar de todos os 6 nós fazerem parte do

mesmo *cluster*, ainda assim é necessário utilizar 2 sistemas de arquivos compartilhados diferentes, pois os nós apenas compartilham arquivos entre os nós de um mesmo provedor de nuvem.

Para realizar a implantação dos contêineres foi utilizado um arquivo de composição para personalizar todas as configurações dos serviços. As configurações desses arquivos, escrito na linguagem YAML, são apresentadas no Código 1. Esse arquivo possui alguns comandos essenciais, tal como o *version*, que indica a versão do *Docker Compose* utilizada. Este comando é importante pois alguns comandos como os de *networks* e volumes são diferentes, dependendo da versão utilizada. Já o comando *services*, que indica o nome do serviço, é importante porque após o nome do serviço são informados os comandos referentes à imagem, volumes e portas utilizadas pelo contêiner daquele serviço.

```
version: "3.7"
services:
    web:
        image: nginx}
        volumes:
            - public:/usr/share/nginx/html:ro
        ports:
            - "80:80"
        networks:
            - webserver
        deploy:
            replicas: 6
            update_config:
                parallelism: 2
                delay: 10s
                order: start-first
            restart_policy:
                condition: on-failure
                max_attempts: 3
                delay: 10s
            resources:
                limits:
                    cpus: "0.50"
                    memory: 512M
    api:
        image: tomcat:8.5.54-jdk8-openjdk
        volumes:
            - api:/usr/local/tomcat/webapps/
        ports:
            - "8080:8080"
        networks:
            - webservice
        deploy:
            replicas: 6
            update_config:
                parallelism: 2
                delay: 10s
                order: start-first
            restart_policy:
                condition: on-failure
                max_attempts: 3
                delay: 10s
            resources:
                limits:
                    cpus: "0.50"
                    memory: 512M
volumes:
    public:
        external: true
    api:
        external: true
networks:
    webserver:
    webservice:
```

Código 1. Arquivo de composição escrito na linguagem YAML.

A utilização de redundâncias é algo essencial quando se utiliza contêineres, pois assim é possível garantir que os serviços estejam sempre disponíveis. Na infraestrutura proposta neste trabalho, os contêineres com os mesmos serviços foram replicados em todos os nós do *cluster*. Em caso de falha de um dos contêineres, após 10 segundos outro é provisionado assumindo seu lugar. A infraestrutura proposta também conta com políticas de atualização. Quando o comando de atualização é executado, 2 novos contêineres com a nova versão do serviço são ativados e os contêineres com a versão antiga são desligados. Desta forma, é possível garantir que os serviços estejam sempre disponíveis mesmo na

presença de falhas ou em procedimentos de atualização.

5. Considerações Finais

A mudança de paradigma no desenvolvimento de *software* evidencia a necessidade de uma gestão eficiente de recursos computacionais na qual a utilização de tecnologias de computação em nuvem e as tecnologias de contêineres se tornam grandes aliadas nesse quesito. Logo, a utilização dessas tecnologias, em conjunto, possibilitam implantação e execução de sistemas distribuídos com maior facilidade e flexibilidade.

Neste artigo foi investigado como prover um melhor gerenciamento de recursos para sistemas de *Smart Farming* que utilizam Internet das Coisas. A utilização de contêineres em nuvem se mostrou adequada para a gestão desses recursos. A conteinerização se torna cada vez mais popular, pois os contêineres são flexíveis, leves, seguros, escaláveis, portáteis e possuem baixo acoplamento. Entre as plataformas de contêineres, o *Docker* se destaca por possuir uma API funcional que permite criar uma solução completa para criação e distribuição de contêineres.

Com a utilização do *Docker* e com os comando *Docker Swarm* é possível criar *clusters* em nuvem de forma simples e rápida. Neste trabalho foi utilizado o *Docker* para criar um *cluster* multi-nuvem nas nuvens da AWS e da *Azure*. Esse *cluster* foi usado como infraestrutura para a implantação de 2 serviços, sendo um *Web Service* e uma aplicação *web* para o gerenciamento de botijões criogênicos.

Sendo assim, a infraestrutura proposta, tem como principal vantagem a sua flexibilidade de escalabilidade, uma vez que podemos expandi-la ou diminuí-la conforme necessário. Outra vantagem é a sua alta disponibilidade devido a redundância de nuvem e de contêineres com os serviços. Como trabalho futuro, novos experimentos serão realizados em uma escala maior, inclusive usando outros provedores de nuvem para uma comparação com outros trabalhos da literatura.

Referências

Amazon (2020). Amazon web services (aws) - cloud computing services. <http://aws.amazon.com>, Acesso em: 8 de Junho de 2020.

Apache (2020). Apache mesos. <http://mesos.apache.org/>, Acesso em: 8 de Junho de 2020.

Bertolino, A., Calabrò, A., Di Giandomenico, F., Nostro, N., Inverardi, P., and Spalazzese, R. (2011). On-the-fly dependable mediation between heterogeneous networked systems. In *International Conference on Software and Data Technologies*, pages 20–37. Springer.

Bhange, M. and Hingoliwala, H. (2015). Smart farming: Pomegranate disease detection using image processing. *Procedia Computer Science*, 58:280–288.

da Cruz, S. A. B., Speranza, E. A., and Yano, I. H. (2018). Uso de programas geoestatísticos no tratamento de grande volume de dados. In *Embrapa Informática Agropecuária*. In: SIMPÓSIO DE GEOTECNOLOGIAS NO PANTANAL.

Docker (2020). Empowering app development for developers — docker. <https://docker.com/>, Acesso em: 8 de Junho de 2020.

Duft, D. (2018). Você sabe a diferença entre smart farming e iot? <https://www.inteliagro.com.br/voce-sabe-diferenca-entre-smart-farming-e-iot/>, Acesso em: 8 de Junho de 2020.

Gomes, R. (2017). *Implantação eficiente de múltiplas coreografias de serviços em nuvens híbridas*. PhD thesis, Universidade de Federal de Goiás, Brasil.

Google (2020). Google cloud: Serviços de computação em nuvem. <cloud.google.com/>, Acesso em: 8 de Junho de 2020.

Initiative, O. C. (2020). Open container initiatives. <https://opencontainers.org/>, Acesso em: 8 de Junho de 2020.

Microsoft (2020). Microsoft azure: Serviços de computação em nuvem. <https://azure.microsoft.com/>, Acesso em: 8 de Junho de 2020.

Mouat, A. (2015). *Using Docker: Developing and Deploying Software with Containers*. "O'Reilly Media, Inc.".

Nginx (2020). Nginx news. <https://nginx.org/>, Acesso em: 8 de Junho de 2020.

Redhat (2020). O que é kubernetes. <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes>, Acesso em: 8 de Junho de 2020.

Rouse, M. (2016). Docker swarm. <https://searchitoperations.techtarget.com/definition/Docker-Swarm>, Acesso em: 8 de Junho de 2020.

Santos, B. P., Silva, L. A., Celes, C., Borges, J. B., Neto, B. S. P., Vieira, M. A. M., Vieira, L. F. M., Goussevskaia, O. N., and Loureiro, A. (2016). Internet das coisas: da teoria à prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 31.

Trindade, L. V. P. (2018). Análise do desempenho da virtualização leve para ambientes com edge computing baseada em nfv.

Virk, A. L., Noor, M. A., Fiaz, S., Hussain, S., Hussain, H. A., Rehman, M., Ahsan, M., and Ma, W. (2020). Smart farming: An overview. In *Smart Village Technology*, pages 191–201. Springer.