Revista do *Fórum Gerencial*, v. 1, n. 2, 2021
© Centro Universitário de Patos de Minas
https://revistas.unipam.edu.br/index.php/forumgerencial

UTILIZAÇÃO E ORQUESTRAÇÃO DE CONTAINERS EM APLICAÇÕES WEB

João Antônio Caetano Rosa[1]
José dos Reis Mota[2]

RESUMO: O presente artigo tem por objetivo principal abordar a utilização e orquestração de *containers* em aplicações *web*, assim como utilizar boas práticas e demonstrar técnicas de observalidade aplicada nessa estrutura, visto que isso pode ser benéfico para empresas de soluções de TI no gerenciamento de suas aplicações. Neste estudo buscou-se analisar as principais técnicas de orquestração, garantia de boas práticas, escalabildiade e a utilização da observabilidade no ambiente da aplicação. Utilizou-se, no decorrer deste trabalho, o método organizacional kanbam que visa aumentar a produtividade e otimizar a gestão do trabalho. Ressalta-se a importância da observabilidade e da maneira como os serviços interagem uns com os outros durante a execução, sendo preciso esse processo ser monitorado, gerenciado e controlado, isto começa com a capacidade de observação e a capacidade de entender o comportamento de arquiteturas de serviços distribuídos. Num âmbito geral buscou-se demonstrar que orquestração de *containers* em aplicações *web*, baseadas em serviços distribuídos e utilizada de maneira correta pode trazer vários benefícios para a organização.
PALAVRAS-CHAVE: Container. Orquestração. Observabilidade. Serviços Distribuídos.

ABSTRACT: The main objective of this article is to address the use and orchestration of containers in web applications, as well as to use good practices and demonstrate techniques of observability applied in this structure, since this can be beneficial for IT solution companies in the management of their applications. This study sought to analyze the main orchestration techniques, guarantee of good practices, scalability and the use of observability in the application environment. Use, in the course of this work, the kanbam organizational method that aims to increase productivity and optimize work management. The importance of observability and the way services interact with each other during execution is emphasized. This process needs to be monitored, managed and controlled. This starts with the observation capacity and the ability to understand the behavior of architectures. distributed services. In a general scope, we tried to demonstrate that container orchestration in web applications, based on diversified services and used correctly, can bring several benefits to an organization.
KEYWORDS: Container. Orchestration. Observability. Distributed Services.

1 INTRODUÇÃO

A evolução das TICS (tecnologia de informação e comunicação) possibilitou à sociedade ter acesso a milhares de informações e complexidade de contextos, próximos ou distantes de sua realidade. Sendo assim, além da criação de padrões de projetos que auxiliam na utilização de padrões comuns para resolver problemas parecidos, surge a necessidade de criar ferramentas que auxiliam o processo de desenvolvimento com a

[1] Aluno de Sistemas de Informação, UNIPAM. E-mail: joaoantonio@unipam.edu.com.br.
[2] Mestre em Ciência da Computação, UFU. E-mail: josereis@unipam.edu.br.
[1]Aluno de Sistemas de Informações, UNIPAM, raphaelcustodio94@gmail.com

[2] Mestre em Redes de Computadores, UFU, henaldobarros@gmail.com

finalidade de acompanhar essa evolução onde, desde a linha zero até o *deploy*, existem vários processos que podem impactar na qualidade do *software* que está sendo criado.

Quando se fala em aplicações *web*, normalmente a primeira coisa que se pensa é em um site, mas o que há por trás dessa aplicação, como ela funciona e como ela está ali disponível para o usuário, na maioria das vezes é algo bastante abstrato quando não se conhece o contexto do que se está sendo visualizado.

O processo de desenvolvimento de uma aplicação *web* é algo que vem gradativamente evoluindo ao longo dos anos e, como o processo evolui, as ferramentas também evoluem e o contexto torna-se cada vez mais abstrato, uma vez que "recriar a roda" é desnecessário, mesmo que muitas vezes seja preciso adaptar pequenas coisas para atender a distintas necessidades.

Durante muito tempo vem se discutindo a melhor maneira de se manter aplicações em funcionamento, várias teses são defendidas e algumas delas se destacam, com objetivo de atender ao máximo os requisitos que garantam a menor taxa de falha e maior excelência possível em todo o ciclo de vida da aplicação, uma vez que, várias maneiras e padrões são aceitos, tornando isso em algo simples ou extremamente complexo dependendo de como é aplicado.

Partindo de um ponto de vista cronológico, a execução do compilador ou interpretador na máquina *host*, até a criação de *containers*, mostra como a escala evolutiva de processos tecnológicos tornou toda a abstração do trabalho de manter a aplicação disponível mais simples, ou até mesmo complexo, ao mesmo tempo que facilita muita a vida de quem trabalha com isso no dia a dia. É claro que requisitos técnicos são necessários para se manter toda essa estrutura, porém a facilidade que essas tecnologias trouxeram e a maneira como agregam valor, tanto para a empresa quanto para os colaboradores de forma mais ágil, se torna inestimável e muitas vezes incalculável.

Quando se refere ao conceito "custo" entende-se como sendo despesas ou contas por exemplo. Entretanto, o custo pode ser visto de várias outras formas, como tempo, desgaste mental, e até mesmo recurso humano, uma vez que a criação e manutenção de processos malfeitos aumentam a carga de material humano para se garantir a vivacidade do produto de *software* da empresa.

Princípios como estabilidade, agilidade, manutenção, escalabilidade, dentre outros vários requisitos não funcionais, vem buscando cada vez mais facilitar todo o processo e ciclo de vida do projeto, tornando-o mais estável e padronizado.

Nessa perspectiva, o presente trabalho tem por objetivo principal abordar a utilização e orquestração de *containers* em aplicação *web*. Considerando o objetivo geral, os objetivos específicos são: estruturar aplicações *web* em *containers*, aplicar boas práticas de provisionamento de infraestrutura como código, buscando escalabilide e a observabilidade em serviços distribuídos que utilizam *containers* orquestrados e gerenciados por plataformas de *cloud computing*.

A confecção deste estudo justifica-se por serem temas atuais e com alta demanda de mercado, já que grande parte das empresas de TI buscam aplicar essas características em suas aplicações.

A tecnologia de *containers* para a virtualização do ambiente de aplicações web pode ser algo muito positivo, pois gera uma economia do ponto de vista

computacional, uma vez que não é necessário a abstração de todo um ambiente de sistema operacional para a execução da aplicação e sim só os requisitos operacionais básicos, como o compilador ou interpretador e as dependências aplicadas para que aquele serviço funcione, de forma que tendo-se uma estrutura criada da maneira correta e seguindo boas práticas, é possível uma evolução gradual num ponto de vista de análise de desempenho e observabilidade de métricas da aplicação, que auxiliam tanto na tomada de decisão em nível técnico como também num ponto de vista de negócio.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentadas as principais ideias e resultados de outros autores que pesquisaram tópicos diretamente associados ao tema do artigo, conceitos relacionados a aplicações *web*, estruturas de serviços distribuídos, aplicações monolíticas e observabilidade de sistemas. Para melhor organização do estudo, a abordagem será estruturada em subseções. 2.1 refere-se à estrutura e utilização de *containers* para ambos os casos; 2.2. refere-se a uma discussão sobre a utilização e orquestração de *containers* em aplicações *web*; 2.3. trata do tema de orquestradores e *cloud computing* para garantia de vários requisitos não funcionais que podem impactar na qualidade do projeto; 2.4 aborda sobre o conceito de observabilidade e seus três pilares buscando explicar sua importância em arquiteturas de serviços distribuídos.

2.1 ESTRUTURAS DE APLICAÇÕES *WEB* E *CONTAINERS*

Uma aplicação *web* pode ser vista muitas vezes como um *software* voltado para a utilização de usuários que tenham acesso a internet, porém o conceito vai muito além disso. Com a abstração de aplicações monolíticas ou a utilização de arquiteturas baseadas em serviços distribuídos, a complexidade pode aumentar bastante, entretanto toda evolução tem por objetivo uma melhoria, seja ela em processos ou até mesmo qualidade do produto. Se tratando de aplicações monolíticas a primeira coisa que se percebe é que, por mais que sua divisão lógica e o desenvolvimento de seu código visem por algo com o menor acoplamento, é possível notar que durante o processo de implantação ou *deploy* desse projeto, o simples fato de ser um monólito já impacta o funcionamento do serviço, uma vez que qualquer alteração de código influencia todo o processo de entrega e gera uma parada geral em qualquer ambiente o qual ele esteja implantada, seja em desenvolvimento, homologação ou produção, sendo a produção o ambiente mais crítico na maior parte dos casos.

Com o passar dos anos e com algumas lições aprendidas, tornou-se cada vez mais comum a utilização de arquiteturas de software baseadas em serviços distribuídos, buscando, dentre outros atributos, a redução do chamado *down time* durante o *deploy* da aplicação. Os serviços distribuídos baseiam-se em estruturas de *softwares* desejavelmente interdependentes e específicas para cada função, muitas vezes subdivididos em estruturas menores chamadas micro serviços, que geralmente comunicam entre si, visando assim o funcionamento modularizado da aplicação, e reduzindo o acoplamento em nível estrutural não somente de código, mas podendo

abstrair-se também na infraestrutura, tornando-se ideais para serem utilizados em *containers*.

Os chamados *containers* podem ser definidos como a virtualização da aplicação, não sendo necessário a abstração de todo um sistema operacional para que *o software* funcione, o *container* já trata de manter somente o necessário para o seu funcionamento como, a estrutura básica do sistema operacional, o compilador ou interpretador da linguagem que está sendo usada, e as dependências necessárias como bibliotecas de código já utilizadas anteriormente que facilitem o desenvolvimento do código dentre outras coisas, mas tudo sendo o mínimo possível para que o *software* não tenha seu desempenho afetado devido a utilização de estruturas e recursos desnecessários (VITALINO, CASTRO 2016).

2.2 UTILIZAÇÃO E ORQUESTRAÇÃO DE *CONTAINERS* EM APLICAÇÕES *WEB*

O contexto de aplicações web atuais se mostra um cenário bastante propício para a utilização de *containers* para virtualização e *deploy* das aplicações, uma vez que cenários onde se exige cada vez mais escalabilidade, resiliência e outros requisitos não funcionais tornam o "terreno" propício para que *containers* tenham cada vez mais espaço e sua utilização torna-se cada vez mais comum em qualquer âmbito de aplicações web, sejam elas aplicações monolíticas ou micro serviços.

O porquê disso acaba sendo óbvio a cada momento, pois grandes gerências de migrações e mudanças na forma de manter um serviço no ar, mostram que por mais que se tenha um esforço significativo no início, isso acaba sendo bem proveitoso a longo prazo. O mesmo paradigma aplica-se na forma de orquestrar pois, quando uma arquitetura já é construída se pensando não somente no seu modelo de aplicação, sendo ela monolítica ou de microsserviços, mas também em sua infraestrutura, isso pode acabar eliminando vários problemas e reduzindo riscos, ao longo da vida do *software* que está sendo trabalhado (CAMISSO, 2020).

O ciclo de vida de uma aplicação *web*, dentro deste contexto, depende de vários fatores como, requisitos de produto ou até mesmo método de desenvolvimento bem ou mal implementados, que podem acarretar numa futura refatoração do código, o que acaba sendo custoso quando o processo é mal planejado. Partindo do pressuposto de que recriar a roda é algo completamente desnecessário, a utilização da tecnologia de *containers,* mostra que o que já foi aprendido até hoje pode se tornar cada vez mais útil e aprimorável, visto que apenas utilizar tecnologia não é garantia que se está fazendo inovação da forma correta pois a principal meta é entregar valor no que se está sendo proposto.

A decisão de utilizar um orquestrador passa sempre por um processo avaliativo, sendo que o contexto em que o mesmo será utilizado pode ser o fator decisivo para a sua implementação, pois tudo que será feito deve ter um propósito e um significado específico para atingir as metas desejadas e entregar o máximo de valor possível, com o melhor custo benefício, e é por isso que o *docker-swarm* acabou sendo deixado de lado e a utilização do *kubernetes* se sobressai ao mesmo (SANTOS, 2020).

2.3 ORQUESTRADORES DE *CONTAINERS* E *CLOUD COMPUTING*

Não dá para se falar em orquestração de *container* sem se falar de *frameworks* como *Kubernetes* ou *Docker-Swarm*, que são as principais e mais conhecidas plataformas de orquestração de *containers* da atualidade, uma vez que se tratando não somente de *Cloud Computing* a utilização dessas poderosas ferramentas podem também ser utilizados em ambientes de servidores *On Promisse* sendo sempre uma ótima opção quando se fala em escalabilidade e resiliência, uma vez que as mesmas foram pensadas para resolver problemas como esses e outros mais.

Partindo do princípio de *Cloud Computing*, é possível observar que desde que grandes empresas como *Google*, *Amazon* e *Microsoft*, começaram a oferecer este tipo de opção, a disputa por quem oferece o melhor serviço se tornou cada vez mais acirrada e o *Kubernetes*, que foi uma iniciativa do *Google* e hoje pertence a *Cloud Native Foundation*, se tornou o que é chamado de a nuvem dentro da nuvem. Isso por que esta poderosa ferramenta é capaz de não somente realizar a orquestração de *containers*, como também a gerência de servidores, agindo quase que como uma camada abaixo do sistema operacional gerenciando todo o *host* da aplicação (SANTOS, 2020).

Por mais que o *Docker-Swarm* seja um bom orquestrador, o *Kubernetes* está a anos luz na sua frente, como pode-se perceber ao analisar estatísticas do site *"dzone"* onde o *kubernetes* supera o *Docker-Swarm* não só em comparações de desempenho como também em pesquisas e utilização por maior parte das empresas de tecnologia, e é por isso que ele será o orquestrador mais citado e utilizado para a experimentação e desenvolvimento deste trabalho (DZONE, 2020).

2.4 OBSERVABILIDADE

Para se compreender melhor sobre o que é observabilidade, primeiro é preciso deixar bem claro o que não é observabilidade.

A observabilidade não é uma caixa pronta, não é um produto que a empresa compra ou desenvolve para "monitorar" o ciclo de trabalho de suas aplicações, a observabilidade é algo que deve ser construído, a aplicação precisa atender a uma série de requisitos para se tornar observável da maneira correta, e isso é atingido através de uma estrutura arquitetural feita de forma inteligente, que permita que a mesma atinja esses requisitos e venha a se tornar observável da forma desejada (GOMES, 2020).

De forma clara e objetiva a observabilidade pode ser descrita como a capacidade de observar os estados anteriores e o estado atual da aplicação, monitorar e detectar falhas eventuais em qualquer ponto do sistema, tendo como objetivo principal não só a manutenção corretiva, como também, a análise da causa raiz da falha, com o objetivo de impedir que esse mesmo erro ocorra novamente não só naquele ponto, mas em qualquer outro dentro do ciclo de trabalho de aplicação (SRIDHARAN, 2020).

Dentro da observabilidade existem três pilares muito bem definidos que constituem todo o conceito por trás da mesma, são eles, *logs*, *metrics* e *tracing*.

2.4.1. *Logs*

Logs são definidos como um conjunto de registros que informam os atuais eventos da aplicação, de modo que os desenvolvedores e até mesmo o pessoal de operação, possam detectar possíveis falhas ou avisos indesejados que estejam ocorrendo no momento atual da visualização e leitura dos mesmos.

Os *logs* são importantes pois na maioria dos casos eles são cruciais para a se saber o que pode ter ocasionado o erro que fez o sistema parar ou não funcionar da maneira desejada naquele momento. Os logs são a consulta direta à informação que o serviço expõe, mostrando o que aconteceu no exato momento da falha e qual foi a tratativa de exceção lançada pelo sistema durante o evento (PRONSCHINSKE, 2020).

2.4.2. *Metrics*

Metrics (métricas) são a coleta de informações de qualquer estado anterior ou atual da aplicação no nível não só de consumo de recursos computacionais, como também de negócio, sendo peças chave e essenciais para a tomada de decisão dentro da organização, servindo como painel de observação de como se dá o comportamento cronológico de todos os estados do serviço.

É necessário saber como utilizar as métricas de forma correta, nem em falta nem em excesso, pois as mesmas são uma das ferramentas que possibilitam à empresa tomar as melhores decisões não só técnicas como de negócio, sendo possível decidir para qual rumo a organização deve seguir a partir de cada análise (PRONSCHINSKE, 2020).

2.4.3 *Tracing*

Dentre os três pilares, o *Tracing* talvez seja o mais "difícil" de se aplicar, pois, o mesmo veio se tornando cada vez mais necessário em arquiteturas de serviços distribuídos, uma vez que é através dele que é possível rastrear toda a rota que a requisição de falha tomou, sendo possível saber qual foi o ponto exato que disparou o alerta dentro do fluxo da malha de serviços.

A abstração de malha se dá pelo fato de que o *tracing*, utiliza a mesma dentro da camada de serviços, permitindo um rastreamento único e identificável dentro do próprio ambiente da aplicação, adicionando um índice no cabeçalho da requisição como um identificador único para cada uma delas, ou até mesmo um rastreamento de ponta a ponta dentro de toda a malha de serviços (PRONSCHINSKE, 2020).

Saber o ponto exato da falha é essencial, e garante não só a rápida correção, como também uma análise bem mais precisa para a prevenção e previsão de possíveis incidentes que venham a ocorrer devido ao mesmo motivo, ou por fluxos de trabalho parecidos.

3 METODOLOGIA

O presente trabalho foi baseado na análise do uso de boas práticas, para a criação de ambientes preparados para serem utilizados por orquestradores de *containers* e aplicando técnicas de observabilidade de sistemas. O KANBAN, que faz parte das metodologias ágeis, foi utilizado para auxiliar na execução de tarefas e gerência do processo de desenvolvimento deste trabalho.

O KANBAN baseia-se na criação de tarefas com datas pré-dispostas e entrega de resultados após a execução de cada tarefa, agregando assim continuamente valor e agilidade em todo o processo de desenvolvimento (ABRANTES, 2020). As principais tecnologias e ferramentas utilizadas para a execução do projeto foram as seguintes:

- ***Kubernetes:*** "*Kubernetes* (K8s) é um produto *Open Source* utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em *container*" (KUBERNETES, 2020).
- ***Docker:*** *Docker* é uma tecnologia de *containers open source* que visa virtualizar o ambiente de aplicação e isolamento de recursos, evitando assim a sobrecarga e utilização de máquinas virtuais na abstração de todo um sistema operacional para execução do *software* (DOCKER, 2020).
- ***Git:*** "Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência" (GIT, 2020).
- ***Gpc:*** O *Google Cloud Platform* ou *GPC* é a plataforma do *Google* que oferece uma gama de serviços baseada em computação em nuvem (GCLOUD, 2020).
- ***Gke:*** "Serviço seguro e gerenciado do *Kubernetes* com suporte de escalonamento automático de quatro direções e *multicluster*" (GKE, 2020).
- ***Terraform:*** "*Terraform* é uma ferramenta para criar, alterar e criar versões de infraestrutura com segurança e eficiência. *Terraform* pode gerenciar provedores de serviços existentes e populares, bem como soluções internas personalizadas" (TERRAFORM, 2020).
- ***Istio:*** "O *Istio* é uma plataforma *open source* de criação e gerenciamento de *service mesh*, permitindo conectar, proteger, controlar e observar toda a malha de serviços e comunicação dos mesmos dentro de um *cluster*" (ISTIO, 2020).
- ***Jaeger:*** "Jaeger é um sistema de rastreamento distribuído lançado como código aberto pela *Uber Technologies*. Ele é usado para monitorar e solucionar problemas de sistemas distribuídos baseados em microsserviços" (JAEGER, 2020).
- ***Kiali:*** "*Kiali* é um console de gerenciamento para malha de serviço baseada no *Istio*. Ele fornece painéis, capacidade de observação e permite operar sua malha com recursos robustos de configuração e validação. Ele mostra a estrutura de sua malha de serviço inferindo a topologia de tráfego e exibe a integridade de sua malha. *Kiali* fornece métricas detalhadas, validação poderosa, acesso *Grafana* e forte integração para rastreamento distribuído com Jaeger" (KIALI, 2020).

- ***Grafana:*** "O *Grafana* permite que você consulte, visualize, alerte e entenda suas métricas, independentemente de onde estejam armazenadas. Crie, explore e compartilhe painéis com sua equipe e promova uma cultura baseada em dados" (GRAFANA, 2020).
- ***Prometheus:*** "O Prometheus é um kit de ferramentas de monitoramento e alerta de sistemas de código aberto" (PROMETHEUS, 2020).
- ***Vegeta:*** "Vegeta é uma ferramenta versátil de teste de carga *HTTP* criada a partir da necessidade de perfurar serviços *HTTP* com uma taxa de solicitação constante. Ele pode ser usado como um utilitário de linha de comando e uma biblioteca" (VEGETA, 2020).

4 DESENVOLVIMENTO E RESULTADOS

Seguindo um fluxo de pesquisa e desenvolvimento contínuo, o KANBAN se mostrou extremamente eficiente, uma vez que as *tasks* e o fluxo de trabalho foram se tornando cada vez mais fluidos à medida do tempo. A execução de cada tarefa foi cronologicamente primordial para o desenvolvimento das seguintes, uma vez que a flexibilidade da metodologia permitiu que mesmo que não fosse em ordem cronológica, o desenvolvimento pudesse ser quebrado em partes importantes que constituíram uma pesquisa que proporcionou a cada etapa uma nova descoberta e comprovação das teses, o que pode ser satisfatória para o propósito final do estudo realizado a respeito da tecnologia de *containers* e uso de orquestradores, mostrando-se uma prática fortemente difundida em todo mundo devido aos seus benefícios para todo o âmbito, não só técnico como financeiro, num ponto de vista organizacional e de negócio.

O primeiro passo foi a coleta de dados a respeito da tecnologia de *containers*, permitindo uma melhor compreensão do funcionamento da mesma, sendo possível uma análise do que estaria por vir, à medida que foi se aprofundando o estudo e testes manuais em cima do *docker*, que se mostrou flexível para a entrada de novas possibilidades e ponto de vistas mais interessantes para o funcionamento de aplicações, sendo de fácil gerenciamento para software baseados em arquitetura distribuída e de infraestrutura como código.

Contudo, mesmo o *docker* sendo uma peça chave para o estudo, foi possível observar que ele fazia parte de toda uma escala evolutiva para se garantir os níveis necessários de infraestrutura inteligente e observável, que foram as principais propostas descritas para o desenvolvimento da solução.

Uma tarefa importante e crucial foi a pesquisa e validação de algumas soluções em *cloud* para se ter ideia de qual seria a mais viável para a criação do protótipo, e o *gpc* se mostrou à frente das demais soluções, não só financeiramente, mas também estruturalmente, além de possuir uma versão *trial*. Por possuir os atributos necessários para o desenvolvimento do trabalho, optou-se por utilizar essa plataforma. Vale a pena lembrar que a solução de orquestração de *container* independe da plataforma de *cloud*, uma vez que a abstração de sua arquitetura é flexível e bastante compatível com a maioria delas, o que facilita a criação de *clusters kubernetes* gerenciados pelo próprio serviço provedor de *cloud.*

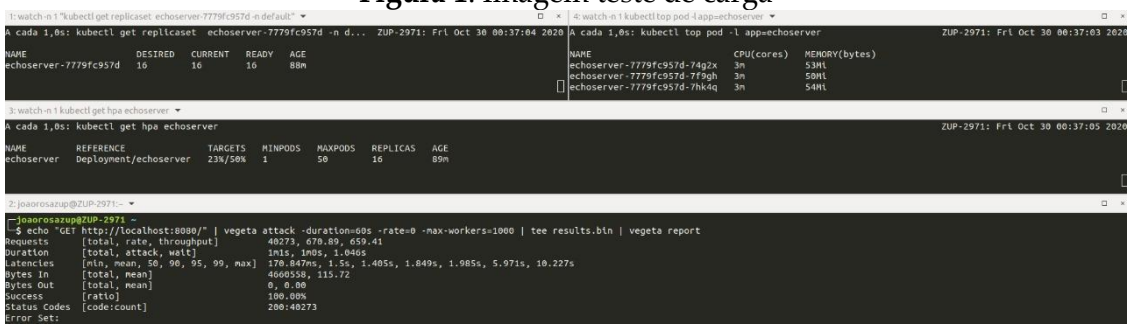
O *kubernetes* mostrou-se mais avançado que qualquer outro serviço de orquestração de *containers*. À medida em que foi sendo aprofundado o estudo desta ferramenta, percebeu-se que suas soluções de automação e gerência são eficientes e fáceis de implementar. Esta etapa foi muito tranquila de ser administrada, pois o *kubernetes* possui uma ampla documentação e comunidade ativa, sendo possível achar milhares de exemplos para cada solução desejada, de acordo com o propósito do estudo.

Após o domínio essencial deste orquestrador, foi a vez de pôr em prática a criação do ambiente, nesse caso do *cluster* de *kubernetes.* O *cluster* foi criado através do *terraform,* sendo usado nesse estudo como a principal ferramenta de provisionamento de infraestrutura como código.

Para que fosse possível testar as soluções propostas por este estudo, foi feito o *deploy* de uma aplicação base conhecida como *echo server*, disponível em "https://github.com/orian/go-echoserver" sendo muitas vezes utilizada para testes de implantação em ambientes de desenvolvimento, o que constitui uma boa prática dentro do universo *Devops*.

Após o *deploy* foi feita a análise do comportamento da aplicação em situações de estresse, através de testes de carga com a ferramenta vegeta para a verificação de escalabilidade comportamental que o software teria no momento que o fluxo de requisições fosse aumentando e exigindo cada vez mais performance e resiliência do mesmo, conforme mostra a Figura 1.

Figura 1: Imagem teste de carga



Fonte: Dados do trabalho, 2020.

Foi possível perceber através da Figura 1 que os testes foram bastante satisfatórios, em relação à performance em um ponto de vista de escalabilidade, pois quando o número de requisições aumentou em um curto espaço de tempo, a sua performance não deixou a desejar, já que através da solução de *HPA(horizontal pod autoscaling)* fornecida pelo *kubernetes,* que é feita através de arquivos de configuração da própria plataforma, a escalabilidade horizontal automática da aplicação se torna uma poderosa aliada para a garantia de resiliência da mesma.

Após a execução desta tarefa foi a vez de criar uma malha de serviços e prover uma rede *mesh* através do *istio.* Seguindo a documentação foi possível verificar que o mesmo possui um binário próprio de instalação e gerência de seus recursos dentro do *cluster*. O *kiali* também foi implementado no *cluster*, sendo uma das *features* que o *istio* traz para a visualização da malha de serviços do ambiente.

Com o intuito de simplificar o processo, todos os recursos presentes no *istio* foram instalados de forma padrão e seguindo por base o passo a passo exposto em sua documentação. Foi realizado também o *deploy* de um conjunto de micro serviços base denominado *Bookinfo*, que o próprio *istio* fornece, disponível em "https://istio.io/latest/docs/examples/bookinfo/". Este serviço foi utilizado para a exemplificação da utilização dos recursos que visam mostrar como funciona a observabilidade dentro do ambiente da aplicação.

O passo seguinte foi a implantação de um recurso chamado *Jaeger* que é o principal responsável por todo o processo de *tracing* das requisições que a aplicação venha a receber. Com ele é possível visualizar todo o caminho que a requisição fez até o exato ponto da falha ou da conclusão, o que torna o processo de depuração em caso de incidentes bem mais fácil de ser realizado, como é possível observar na Figura 2.

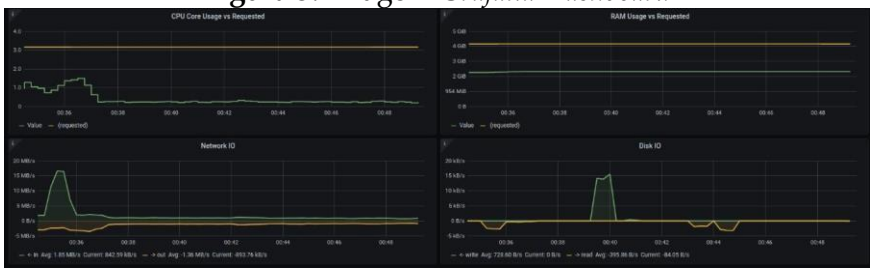
Figura 2: Imagem *Jaeger tracing Dashboard*

Fonte: Dados do trabalho, 2020.

Com o *Jaeger* em funcionamento foi a vez de utilizar o *grafana* como centralizador de *dashboards*, para a visualização das métricas expostas pelos serviços do cluster. O *grafana* foi utilizado localmente, uma vez que para o propósito deste trabalho não foi necessário a criação de um ambiente separado para o mesmo. No entanto, é uma boa prática utilizá-lo fora do ambiente da aplicação, para que a mesma não interfira em seu funcionamento em caso de incidentes.

Para que o *grafana* funcione da maneira correta, é necessário também um serviço de exposição de métricas, pois ele é utilizado somente para que sejam consultados e criados os *dashboards* de monitoramento e possíveis alertas com base nas métricas consumidas. Partindo desse pressuposto, foi a vez de expor o serviço do *prometheus*, que por padrão já vem implantado no cluster de *kubernetes* gerenciado do *gcloud*, onde seguindo as devidas configurações, o *prometheus* passou a exportar para o *grafana* todas as métricas encontradas no ambiente. Logo após conexão entre os dois serviços ser realizada com sucesso, foram criados os respectivos *dashboards* de visualização de métricas, concluindo assim parte da *stack* de observabilidade, como mostra a Figura 3.

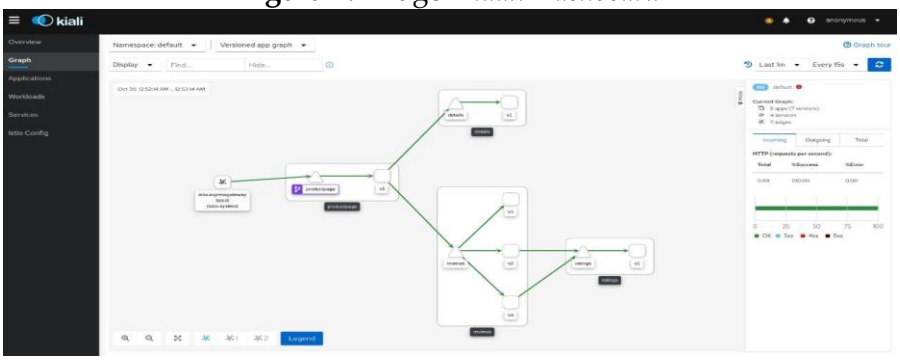
Figura 3: Imagem *Grafana Dashboard*



Fonte: Dados do trabalho, 2020.

Em seguida, foi realizado o teste de funcionamento de toda a *stack* de observabilidade, através de requisições feitas no serviço de Bookinfo, onde foi possível observar no *dashboard* do *kiali*, representado na Figura 4, o fluxo que as requisições seguiram até chegar em seu destino final.

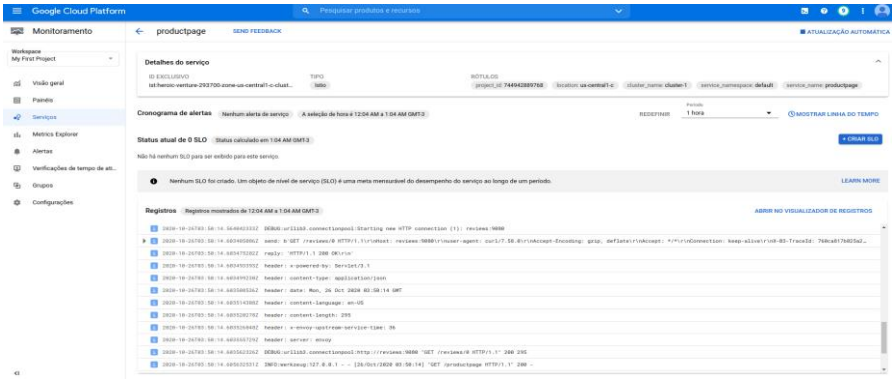
Figura 4: Imagem *kiali Dashboard*



Fonte: Dados do trabalho, 2020.

Depois disso, foi a vez de verificar os *logs* da aplicação, uma boa prática é manter isso centralizado de forma que fique fácil para a equipe de operações agir da forma mais rápida e precisa possível, pois nesses casos é necessário fazer a junção de *logs* de todas as aplicações envolvidas no processo. Na Figura 5 está representada a utilização do serviço de *StackDriver* do *Google* para a centralização e visualização de *logs* dos serviços.

Figura 5: Imagem *logs Stackdriver*



Fonte: Dados do trabalho, 2020.

5 CONCLUSÃO

O presente trabalho teve por objetivo abordar as principais técnicas e boas práticas relacionadas à utilização e orquestração de *containers* em aplicações *web*, com foco voltado para arquitetura de serviços distribuídos, buscando demonstrar a resiliência dos serviços através de testes de carga e abordar a importância da observabilidade aplicada de forma correta, sendo usada para garantir o monitoramento e tomada de decisão em nível técnico e de negócio.

Com base nos estudos apresentados, é possível se ter uma ideia de qual direção tomar para se implementar da melhor forma possível a tecnologia de *containers* orquestrados na infraestrutura de aplicações. Espera-se que esse estudo possa contribuir com desenvolvedores e empresas que visam uma melhoria contínua em seus processos de implementação de infraestruturas inteligentes e auto gerenciadas, onde a aplicação da observabilidade pode garantir vários benefícios antes não explorados.

O trabalho teve grande parte de seus objetivos alcançados, uma vez que o uso de *softwares* bases para testes, muitas vezes não abrangem todos os casos de uso que uma aplicação empresarial e complexa venha a exigir para se garantir a excelência específica de todos os seus pontos. Porém, os mais importantes, ou grande parte deles, puderam ser testados. Comprovou-se que, as técnicas utilizadas garantem que a maioria dos requisitos não funcionais de aplicações baseadas em *containers* e serviços distribuídos venham a necessitar puderam ser alcançados.

Ao longo dos anos as pesquisas sobre utilização e orquestração de *containers* vem evoluindo cada vez mais. Uma enorme gama de ferramentas e técnicas vem surgindo a cada dia, o que torna essa área de estudo bastante atrativa para vários estudiosos e entusiastas de tecnologia. Tendo isso como base, o estudo pode ser uma boa referência para quem desejar empregar essas técnicas em seus processos de desenvolvimento de software, na criação de aplicações baseadas em *containers* com arquitetura de serviços distribuídos e infraestruturas provisionadas como código.

REFERÊNCIAS

ABRANTES, Letícia. **Kanban**: entenda o que é e como funciona o método Kanban. 2018. Disponível em: https://rockcontent.com/blog/kanban/. Acesso em: 20 maio 2020.

CAMISSO, Jamon; JETHA, Hanif; JUELL, Kathleen. Kubernetes For Full-Stack Developers. **DigitalOcean**, New York City, New York, USA. 2020.

DOCKER. **About Docker**. Disponível em:https://www.docker.com/company/. Acesso em: 28 junho 2020.

DZONE. **"Docker Swarm ou Kubernetes?"**: é a pergunta certa a se fazer?. Disponível em: https://dzone.com/articles/quotdocker-swarm-or-kubernetesquot-is-it-the-righ/. Acesso em: 28 jun. 2020.

GCLOUD. **Ofereça mais soluções com o Google Cloud**. Disponível em: https://cloud.google.com/. Acesso em: 22 maio 2020.

GIT. **Sistema de controle de versão distribuído**. Disponível em: https://git-scm.com/. Acesso em: 25 maio 2020.

GKE. **Google Kubernetes Engine**. Disponível em: https://cloud.google.com/kubernetes-engine?hl=pt-br/. Acesso em: 25 maio 2020.

GOMES, Pedro César Tebaldi. **O que é observabilidade e qual a diferença para a monitoração?**. Disponível em: https://www.opservices.com.br/observabilidade/. Acesso em: 30 junho 2020.

GRAFANA. **A plataforma de análise para todas as suas métricas**. Disponível em: https://grafana.com/grafana/. Acesso em: 26 junho 2020.

ISTIO. **What is Istio?**. Disponível em: https://istio.io/latest/docs/concepts/what-is-istio/. Acesso em: 25 junho 2020.

JAEGER. **Jaeger**: open source, end-to-end distributed tracing. Disponível em: https://www.jaegertracing.io/. Acesso em: 30 junho 2020.

KIALI. **Service mesh managementfor Istio**. Disponível em: https://kiali.io/. Acesso em: 30 junho 2020.

KUBERNETES. **Orquestração de containers prontos para produção**. Disponível em: https://kubernetes.io/pt/. Acesso em: 20 maio 2020.

PROMETHEUS. **What is Prometheus?**. Disponível em: https://prometheus.io/docs/introduction/overview/#what-is-prometheus/. Acesso em: 30 junho 2020.

PRONSCHINSKE, Mitch. **Monitoring demystified**: a guide for logging, tracing, metrics. Disponível em: https://techbeacon.com/enterprise-it/monitoring-demystified-guide-logging-tracing-metrics/. Acesso em: 25 ago. 2020.

SANTOS, Lucas. **Kubernetes**: tudo sobre orquestração de containers. Casa do Código, Rua Vergueiro, 3185, 8º andar, São Paulo, SP, Brasil. 2020.

SRIDHARAN, Cindy. **Distributed systems observability**: a guiding to building robusted systems. O'reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. maio 2018.

TERRAFORM. **O que é Terraform?**. Disponível em: https://www.terraform.io/intro/index.html/. Acesso em: 30 jun. 2020.

VEGETA. **Vegeta**. Disponível em: https://github.com/tsenart/vegeta/. Acesso em: 30 jun. 2020.

VITALINO, J. F. N.; CASTRO, M. A. N. **Descomplicando o Docker**. Brasport Livros e Multimídia LTDA., Rua Pardal Mallet, 23, Tijuca, Rio de Janeiro, Brasil. 2016.