Luiz Fernando Altran

Orquestração personalizada de contêineres

Cascavel-PR

2022

Luiz Fernando Altran

Orquestração personalizada de contêineres

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel

Centro de Ciências Exatas e Tecnológicas – CCET

Programa de Pós-Graduação em Ciência da Computação – PPGComp

Orientador: Dr. Guilherme Galante

Cascavel-PR

2022

Luiz Fernando Altran
Orquestração personalizada de contêineres/ Luiz Fernando Altran. – Cascavel-PR, 2022-
102p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Guilherme Galante

Dissertação (Mestrado)– Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel
Centro de Ciências Exatas e Tecnológicas – CCET
Programa de Pós-Graduação em Ciência da Computação – PPGComp, 2022.

1. *multi-cloud*. 2. contêineres. 3. escalonador. 4. *Kubernetes*.

Luiz Fernando Altran

Orquestração personalizada de contêineres

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Trabalho aprovado. Cascavel-PR, 17 de fevereiro de 2022:

Dr. Guilherme Galante
Orientador(a)

Dr. Darlon Vasata
Instituto Federal do Paraná

Dr. Edson Tavares de Camargo
Universidade Tecnológica Federal do Paraná

Dr. Marcio Seiji Oyamada
Universidade Estadual do Oeste do Paraná

Cascavel-PR
2022

Este trabalho é dedicado à minha esposa Ana Célia,
por todo apoio e suporte prestado na minha jornada científica,
principalmente nos momentos de maiores dificuldades e
desafios enfrentados ao longo do percurso.

Agradecimentos

Decidir pelo ingresso e início em um programa de estudos como foi o *PPGComp Unioeste* requer muito mais que vontade, precisa de incentivo e apoio de quem confia e acredita no seu potencial. A partir do momento em que você já possui uma bagagem de conhecimento oriunda de graduações ao longo da vida e experiência profissional na área de atuação, se torna cada vez mais difícil optar por dedicar tempo em mais uma jornada, e nestes momentos, o apoio de amigos e familiares se torna peça chave.

Neste processo, meu amigo *Ernanny Figueiredo* teve papel fundamental, ao decidirmos por ingressar juntos para obter um apoio mútuo, não apenas com foco no compartilhamento de ideias, mas incentivo ao longo dos desafios que seriam enfrentados, e sem dúvidas isso fez todo o diferencial.

Despertar as habilidades científicas adormecidas em cada um de nós também não é uma tarefa fácil, requer empenho e persuasão, e meu orientador, o professor Dr. Guilherme Galante foi incrível, além do fato de acreditar na proposta inicial e objeto alvo da pesquisa, e também por compreender minhas restrições pessoais envolvendo a conciliação com atividades profissionais e disponibilidade de tempo.

A todos os envolvidos no programa *PPGComp Unioeste*, em especial ao professor Dr. Luiz Antonio Rodrigues, pelo empenho e dedicação em ofertar educação do mais alto nível na área de tecnologia, composto por um quadro de docentes altamente qualificado e competente.

Esta pesquisa contou com o apoio de programas voltados para apoio e incentivo no uso de computação em nuvem, com destaque à *AWS Educate*[1], oferecido pela *Amazon*, *Azure for Students*[2], oferecido pela *Microsoft* e especialmente à *Oracle*, com o *Oracle for Research*[3], onde obtive grande apoio e suporte com a concessão de créditos de forma substancial, além de acompanhamento multidisciplinar durante as etapas de implementação do escalonador personalizado, possibilitando a execução de todos os experimentos em um cenário real de operação.

[1] <https://aws.amazon.com/education/awseducate>
[2] <https://azure.microsoft.com/en-us/free/students>
[3] <https://www.oracle.com/research>

Resumo

ALTRAN, Luiz Fernando. **Orquestração personalizada de contêineres**. Orientador: Dr. Guilherme Galante. 2022. 102f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

As políticas de alocação de contêineres presentes em orquestradores modernos, tal como o *Kubernetes*, são completamente agnósticas no que diz respeito a demandas específicas das aplicações ou atendimento a requisitos de negócio. Geralmente realizam a alocação das aplicações simplesmente espalhando-as entre os nós de trabalho usando algoritmos como *Round-Robin* ou *First-Fit*. Além disso, ao se delinear o estado da arte, verifica-se que as estratégias propostas não satisfazem os critérios de escalonamento de aplicações em ambientes de produção reais. Neste trabalho apresenta-se uma técnica que permite a personalização do escalonamento como alternativa ao comportamento padrão oferecido pelas ferramentas de orquestração de cargas de trabalho conteinerizadas em ambientes *multi-cloud*, realizando tratativas e validações pertinentes para se atingir o objetivo de realizar o direcionamento das instâncias da aplicação a nós computacionais com maior afinidade. Para isso, são consideradas características desejáveis ou impositivas, obtidas a partir da etapa de levantamento de requisitos durante a concepção da aplicação, ou ainda, na etapa de contratação do serviço de hospedagem em nuvem. Buscando oferecer uma alternativa para este comportamento e num formato de fácil utilização, propõe-se um escalonador personalizado que realiza uma análise de afinidade a partir de rótulos definidos em metadados dos objetos que representam cada um dos nós computacionais e cargas de trabalho em um ambiente orquestrado, e como segunda característica, prioriza a escolha através daqueles nós com a maior capacidade computacional ociosa, garantindo um direcionamento que respeite regras e restrições pré-definidas, de acordo com requisitos de negócio da aplicação. Para validação, foi realizada a construção de cenários hipotéticos com definição de rótulos aleatórios, que de alguma forma possuíam afinidade com um ou mais nós computacionais disponíveis no ecossistema *multi-cloud* construído, constituído por 25 nós distribuídos por 4 fornecedores de nuvem pública, com diferentes configurações de *hardware* e localização geográfica, muito semelhante aquele encontrado em empresas que exploram este tipo de serviço. Também foi realizada uma validação exclusiva para metrificação de desempenho do processo de escalonamento, com o objetivo de analisar as diferenças de tempo gasto entre um escalonador padrão e o proposto, sob as mesmas condições e cargas de trabalho.

Palavras-chave: *multi-cloud*; contêineres; escalonador; *Kubernetes*

Abstract

ALTRAN, Luiz Fernando. **Custom scheduler for Kubernetes**. Orientador: Dr. Guilherme Galante. 2022. 102f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

Container allocation policies present in modern orchestration tools, such as Kubernetes, are completely agnostic with respect to specific application requirements or meeting business rules. They usually perform the schedule of applications simply by spreading them among the worker nodes using algorithms such as Round-Robin or First-Fit. Furthermore, when outlining the state of the art, it appears that the proposed strategies do not satisfy the criteria for scheduling applications in real production environments. This work presents a technique that allows the customization of scheduling as an alternative to the default behavior offered by the orchestration tools of containerized workloads in multi-cloud environments, carrying out pertinent negotiations and validations to achieve the objective of performing the scaling of the application instances to compute nodes with higher affinity. For this, desirable or impositive features are considered, obtained from the requirements phase during the design of the application, or even at the phase of contracting the cloud hosting service. Looking to offer an alternative to this behavior and in an easy-to-use approach, we propose a custom scheduler that performs an affinity analysis from labels defined in metadata of objects that represent each of the compute nodes and workloads in an orchestrated environment, and as a second feature, prioritize the choice through those nodes with the highest idle computational resources, ensure a result that respects pre-defined rules and restrictions, according to the application business requirements. For validation, hypothetical scenarios were built with the definition of random labels, which somehow had an affinity with one or more compute nodes available in the built multi-cloud environment, consisting of 25 nodes distributed across 4 public cloud providers, with different hardware configurations and geographic location, very similar to that found in companies that use this kind of service. An exclusive validation was also carried out to metrify the performance of the scheduling process, in order to analyze the differences in time spent between the default scheduler and the proposed one, under the same conditions and workloads.

Keywords: multi-cloud; container; scheduler; Kubernetes

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Interface de programação de aplicações
AWS	*Amazon Web Services*, nuvem computacional pública
GCP	*Google Cloud Platform*, nuvem computacional pública
GDPR	Legislação de privacidade à dados da União Europeia
IaaS	*Infrastructure as a Service*
LGPD	Lei geral de proteção à dados
LXC	Contêineres Linux
OCI	*Oracle Cloud Infrastructure*, nuvem computacional pública
PaaS	*Platform as a Service*
QoS	*Quality of Service* - Qualidade do Serviço
SaaS	*Software as a Service*
SLA	*Service Level Agreements*
vCPU	Unidade de processamento virtualizada
VM	*Virtual Machine* - Máquina Virtual
YAML	*Yet Another Markup Language*

Sumário

1 Introdução

A exploração de tecnologias de conteinerização tem se difundido cada vez mais, permitindo uma maior portabilidade e escalabilidade das aplicações. Além disso, garantindo um completo isolamento dos recursos necessários para sua execução. Por se tratar de um modelo de virtualização, os contêineres se diferenciam do modelo tradicional pelo fato de estarem localizados na camada dos sistemas operacionais, herdando grande parte dos recursos necessários para sua execução do sistema hospedeiro (NGUYEN et al., 2020).

Nos ecossistemas de computação em nuvem, públicas ou privadas, a adoção dos contêineres tem tido um avanço acelerado, popularizando-se pelas vantagens oferecidas em relação a rápida distribuição de aplicações e melhor utilização dos recursos computacionais. Além disso, com a evolução do conceito de conteinerização, a distribuição de micro-serviços e aplicações tornou-se bastante fácil (MENOUER, 2020).

Processos operacionais relacionados a gestão dos contêineres e construção de ambientes *multi-cloud*[1] foram simplificados com o surgimento de ferramentas especializadas nestas tarefas, pois automatizaram a orquestração de contêineres e tarefas de elasticidade de recursos computacionais, alocação de endereços de IP, segurança, monitoramento, balanceamento de carga e implantação de instâncias em multi-fornecedores e multi-regiões geográficas. Operações de realocação de máquinas virtuais sob demanda, através dos vários fornecedores contratados, possibilitam uma maior eficiência do ambiente *multi-cloud*, favorecendo a redução de custos e maior aproveitamento de oferta dos serviços, baseado nos requisitos de *hardware* necessários para distribuição das aplicações (ZHONG; BUYYA, 2020).

Com o objetivo de automatizar o gerenciamento de aplicações conteinerizadas, diversas ferramentas de orquestração de contêineres e gerenciamento de ambientes *multi-cloud* ganharam notoriedade, como o *Docker Swarm*, *Apache Mesos* e *Kubernetes*. Estas ferramentas trazem consigo uma variedade de componentes e ferramentas para garantir a alta disponibilidade das aplicações e gestão a eventos de escalonamento, de acordo com as demandas e consumo dos recursos disponíveis em um *cluster* computacional. Os escalonadores, contidos nestas ferramentas, possuem comportamento genérico, com foco na disponibilização das aplicações buscando o melhor aproveitamento dos recursos computacionais ociosos (RODRIGUEZ; BUYYA, 2018).

Idealmente, conforme as aplicações conteinerizadas são enviadas para implantação, um sistema de orquestração deve alocar os respectivos contêineres de modo rápido em um dos recursos disponíveis, considerando fatores como a capacidade das máquinas disponíveis,

[1] Utilização de duas ou mais nuvens computacionais, distribuídas entre diferentes fornecedores

custo de implantação, requisitos de desempenho e qualidade de serviço (*QoS*) da aplicação, tolerância a falhas e consumo de energia.

No entanto, as políticas de alocação presentes nos orquestradores atuais (*Kubernetes* e *Docker Swarm*, por exemplo) são completamente agnósticas no que diz respeito a demandas específicas das aplicações ou manutenção da *QoS*. Geralmente utilizam políticas que realizam a alocação das aplicações simplesmente espalhando os contêineres entre os nós de trabalho usando algoritmos como *Round-Robin* ou *First-Fit* (KUBERNETES, 2020b).

Dentro do contexto de escalonamento voltado a demandas específicas, várias propostas e estratégias são sugeridas na literatura. Segundo (MENOUER, 2020), o princípio destas estratégias é o mesmo, escolher dentro de vários nós computacionais que constituem um *cluster* aquele que irá executar a instância da aplicação. A literatura propõe estratégias relacionadas a comportamentos específicos do escalonador, como a escolha de nós que estejam hospedados em fornecedores com determinadas características (baixo nível de emissão de carbono, localização geográfica, arquitetura de *hardware*, etc.), melhor aproveitamento da oferta de serviços por diferentes fornecedores, etc. (BAUR et al., 2018; ROCHA et al., 2019; JAMES; SCHIEN, 2019; ROSSI et al., 2020). Tais soluções são efetivas nos casos onde apenas um desses objetivos é considerado. No entanto, o plano de alocação de contêineres não é tão simples se múltiplos objetivos tiverem que ser considerados simultaneamente. O problema ainda se amplifica em um cenário multi-usuário, no qual cada um deles possui um conjunto de aplicações com requisitos de negócio distintos, e *multi-cloud*, onde diversos provedores são utilizados.

É nesse contexto que o presente trabalho se insere. Ao considerar requisitos que envolvam múltiplos objetivos, multi-usuário e *multi-cloud*, desenvolveu-se uma estratégia flexível para que seja possível realizar o escalonamento de modo apropriado. Assim, a contribuição deste trabalho é apresentar uma solução que permite que o escalonamento de contêineres possa ser feito considerando um conjunto de requisitos de negócio personalizados de uma determinada aplicação ou de seu proprietário. Isso é feito, estendendo-se o esquema de rotulagem, já presente em alguns orquestradores, para permitir a atribuição de características aos nós computacionais e requisitos às aplicações, bem como realizar a vinculação das cargas de trabalho aos nós de maior afinidade.

Essa técnica foi implementada na ferramenta de orquestração *Kubernetes*, de maneira que tais rótulos foram considerados durante as tarefas de escalonamento, sendo necessário para isso a implementação de um componente personalizado, que substitui o comportamento padrão da ferramenta de orquestração, e assim, garante a aplicabilidade dos mesmos. Desta forma, durante a etapa de disponibilização de novas instâncias, os rótulos definidos para as cargas computacionais são confrontados com as definições realizadas nos nós computacionais, garantindo assim que a escolha seja por aquele com a maior pontuação em relação a afinidade.

Vale destacar que esta estratégia não se limita a cenários específicos, mas permite que qualquer tipo de rótulo seja definido, sendo necessário apenas que haja coerência entre estes rótulos utilizados nas cargas de trabalho e nos nós computacionais membros do *cluster*, garantindo assim sua heterogeneidade na construção de estratégias de acordo com o objetivo que se deseja alcançar. Sua utilização também não está restrita ou limitada a ambientes compostos por nós computacionais disponíveis em um único sítio, mas compreende até mesmo *clusters* computacionais distribuídos através de múltiplos fornecedores, tendo como único requisito a característica de todos os nós computacionais que o compõem estarem vinculados à mesma tenância da ferramenta de orquestração.

Para validar a implementação do escalonador personalizado foi construído um ambiente *multi-cloud* composto por nós computacionais de diferentes fornecedores, gerenciados pela ferramenta de orquestração *Kubernetes*, também foi definido um conjunto de experimentos envolvendo cenários de negócio, compostos por restrições às quais restringem a disponibilização de instâncias da aplicação à determinadas características dos nós computacionais e comparativos com abordagens encontradas no estado-da-arte, onde a utilização de rótulos permite que resultados semelhantes e até mesmo equivalentes sejam atingidos. A realização destas simulações comportamentais do escalonador personalizado, teve como apoio aplicações *open-source* de *benchmark* no consumo de recursos de *hardware*, como CPU e memória, que foram submetidas ao escalonador padrão e posteriormente ao escalonador personalizado, e os resultados obtidos entre os dois modelos confrontados.

Os resultados obtidos se demonstraram promissores, onde o modelo personalizado garantiu um melhor controle do nó computacional alvo das instâncias utilizando os rótulos, permitindo que diferentes objetivos fossem alcançados de forma independente entre cada um dos cenários. Ao considerar as implementações de escalonadores personalizados disponíveis no estado-da-arte, concebidos para objetivos específicos, conseguimos portabilizar sua configuração ao modelo de rótulos, destacando sua flexibilidade de aplicação. Em relação ao tempo necessário para realizar o processo como um todo, há uma diferença notadamente justificável, haja vista os novos processos de análise dos rótulos e cálculo de ociosidade dos nós computacionais durante a escolha daquele mais indicado.

O restante do documento está organizado da seguinte forma. O Capítulo 2 apresenta uma visão geral da computação em nuvem, abordando os principais conceitos envolvidos neste modelo de computação. O Capítulo 3 apresenta os aspectos envolvendo a orquestração de contêineres, suas funcionalidades, objetivos e trabalhos acadêmicos na área. No Capítulo 4 são apresentados os detalhes relacionados à implementação e utilização do escalonador personalizado. No Capítulo 5 são apresentados os experimentos e cenários construídos para validação do orquestrador. Por fim, o Capítulo 6 expõe os argumentos e considerações finais, abordando inclusive sugestões de trabalhos futuros.

2 Conceitos Fundamentais

Essa seção tem como objetivo apresentar os principais conceitos relacionados à compreensão deste trabalho. A Seção 2.1 apresenta conceitos e fundamentos da computação em nuvem, suas principais características em relação à oferta de produtos, modelos da prestação de serviços e seus diferentes tipos de classificação. A Seção 2.2 introduz os diferentes tipos de solução *multi-cloud*, incluindo modelos de arquitetura, ferramentas de gerenciamento e critérios para aplicações de usuário explorarem este tipo de ecossistema. Uma visão geral à virtualização computacional e utilização de contêineres para aplicações é apresentado na Seção 2.3, abordando diferenças das duas arquiteturas e destacando os principais pontos de vantagem no uso de contêineres em ambientes *multi-cloud*.

2.1 Cloud Computing

A computação em nuvem nada mais é que um conjunto de computadores, equipamentos de rede e armazenamento, que prestam serviços especializados e que possuem um ambiente para que tais serviços possam ser prestados aos consumidores que desejam contratá-los, de forma fácil e rápida. Estes ambientes podem estar hospedados em uma infraestrutura e administração própria, também conhecida como *on-premise*, ou em um local externo, acessado remotamente. Nestes casos, a infraestrutura é administrada pelos próprios fornecedores, que aliaram sua demanda interna para sustentar seus respectivos modelos de negócio, sua capacidade computacional ociosa, a presença global de seus negócios e seu alto conhecimento técnico, como *Google*, *Microsoft*, *Amazon* e *Oracle*.

De acordo com (MELL; GRANCE, 2011), as características da computação em nuvem são:

- Serviços sob-demanda e *self-service*: A escolha dos serviços que serão contratados, é definida pelo próprio consumidor, que poderá solicitar tais recursos computacionais, de forma autônoma e parametrizando de acordo com sua necessidade;
- Acesso compartilhado a recursos: Os recursos computacionais são fornecidos a múltiplos consumidores utilizando um modelo de *tenant* compartilhado, em que uma única instância do *hardware* ou *software* atende as demandas de vários clientes. Atributos relacionados à privacidade e interdependência de recursos são garantidos neste modelo, de tal forma que um cliente não afete o desempenho dos serviços contratados por outro. Os clientes não possuem um controle exato de onde os serviços estarão hospedados para execução, mas apenas a parametrização em alto nível quanto

a localização de onde desejam que tais recursos sejam disponibilizados, como um país, estado ou *datacenter* (CHANDRASEKARAN, 2014);

- Elasticidade e escalabilidade: O provedor de serviços deve possuir mecanismos para escalonar dinamicamente os recursos de sua infraestrutura, liberando mais carga computacional, ou reduzindo-as, com base nos requisitos das aplicações hospedadas. A realização desta tarefa pode envolver uma parametrização prévia dos serviços contratados, para que por exemplo, execute após o consumo atingir um percentual X durante um tempo N, tanto para expandir quanto para contrair a alocação de recursos. Na visão do consumidor, a capacidade de provisionamento é ilimitada, e recursos podem ser adquiridas em qualquer quantidade e a qualquer momento (CHANDRASEKARAN, 2014);
- Mensuração e monitoramento: O ecossistema deve possuir um conjunto de ferramentas para monitorar o estado atual do sistema de forma geral, o consumo de recursos, como armazenamento, processamento, tráfego de rede, dentre outros. Tais métricas favorecem a transparência na prestação de serviços, tanto para o provedor da nuvem computacional, quanto para o consumidor que contratou o serviço. Costumeiramente, fornecedores de nuvens fornecem painéis gerenciais onde o próprio cliente pode realizar o monitoramento dos serviços em uso.

Os modelos da prestação do serviço em nuvem podem ser ofertados de três formas, sendo que cada um deles possui seus benefícios e aplicações (MALATHI, 2011). O processo de escolha entre cada um deles requer seu entendimento específico.

- *Software as a Service (SaaS)* é o modelo em que os serviços podem ser acessados sem a necessidade de efetuar algum tipo de download, instalação ou configuração local do produto contratado, na forma de um software pronto, e que o próprio fornecedor é responsável por toda a infraestrutura de tecnologia para manter a disponibilidade de tal serviço. Além disso, o fornecedor também assume a responsabilidade pela manutenção e suporte aos usuários da plataforma.
- *Platform as a Service (PaaS)* é o modelo em que o fornecedor do serviço em nuvem disponibiliza um ambiente para que o cliente construa e distribua suas próprias aplicações. Os recursos computacionais utilizados, como, servidores, armazenamento e redes não são gerenciados pelo cliente, mas apenas pela sua aplicação que está em execução neste ecossistema, além disso, também deve ser possível efetuar configurações especificas para o ecossistema do próprio cliente, sem que haja interferência em outros serviços que compartilhem a mesma infraestrutura.
- *Infrastructure as a Service (IaaS)* é o modelo em que o cliente fica responsável por realizar a gestão e provisionamento de recursos computacionais, como: armazenamento,

rede, memória, etc. Neste ambiente, deve ser possível a execução e distribuição de qualquer tipo de sistema, inclusive sistemas operacionais completos, aplicações, bibliotecas e outros recursos necessários pelo seu ecossistema, porém, o controle sobre a infraestrutura computacional (*hardware*) é de responsabilidade do fornecedor. Neste modelo, podem ser considerados o fornecimento de servidores, *storages* e máquinas virtuais.

Em cada um dos modelos de prestação de serviço, é esperado que alguns requisitos sejam atendidos pelo fornecedor, respeitando as restrições e limites de cada um deles. O modelo *SaaS* tem a administração de licenças dos produtos contratados e atributos de rede, o modelo *PaaS* tem a administração de atributos para disponibilização e distribuição de aplicações, e o modelo *IaaS* tem a administração de atributos referente à *hardware* (processamento, memória, armazenamento em disco, interfaces de rede e máquinas virtuais), *software* (sistema operacional e outros pré-instalados), *storage* (capacidade de armazenamento), *network* (especificações para priorização de pacotes, largura de banda e volume tráfego dos dados) e requisitos de alta disponibilidade (planos e rotinas de *backup* de todos os recursos contratados) (CHANDRASEKARAN, 2014).

As nuvens também podem ser classificadas de acordo com o seu modo de implantação, públicas se estiverem amplamente disponíveis para contratação, ou privadas, ao se referirem de estruturas próprias das organizações em relação ao seu ecossistema computacional (também conhecidos como *datacenter on-premise*). A classificação completa destes modelos são:

- **Nuvens privadas** possuem sua infraestrutura provisionada para uso exclusivo de uma única organização, que podem compreender vários clientes, quando esta organização possui filiais, por exemplo. A propriedade, gestão e operação é de responsabilidade da organização, ainda que, a localização física pode ser dentro ou fora de suas instalações;
- **Nuvens comunitárias** possuem uma infraestrutura para uso compartilhado por clientes de organizações que compartilham o mesmo interesse e particularidades, tais como: missão, requisitos de segurança, regras e permissões de acesso. A propriedade, gestão e operação é de responsabilidade de um ou mais participantes desta organização, e assim como no modelo privado, a localização física pode ser dentro ou fora de suas instalações;
- **Nuvens públicas** possuem uma infraestrutura para uso do público em geral, sua propriedade pode ser de uma ou mais empresas, instituições de ensino ou governos, sendo que a localização física está no ambiente do fornecedor;

- **Nuvens híbridas** são a combinação de duas ou mais infraestruturas de computação em nuvem - pública e privada as mais utilizadas - que possuem suas características próprias, mas que, utilizando os mesmos padrões de implementação e tecnologias, fornecem recursos para portabilidade e balanceamento das aplicações entre elas. A adoção deste modelo pelas empresas pode ser justificado para separar os serviços de acordo com a sensibilidade dos dados, reservando seu ecossistema computacional privado para dados críticos ou que possuam uma maior relevância de privacidade, e os demais em nuvens públicas, que podem possuir uma maior latência sem afetar a qualidade de seus serviços.

A escolha do modelo está ligada aos requisitos do técnicos do usuários, a aplicação que será distribuída e o escopo de acessibilidade da mesma, com isso é possível obter um direcionamento a cada um deles. Nuvens privadas tem a finalidade de servir pessoas de uma organização, geralmente estão localizadas em ambientes *on-premise*. Nuvens comunitárias são utilizadas para realizar o compartilhamento entre múltiplas empresas, geralmente ligadas a objetivos em comum, de forma que todos tenham benefícios mútuos ao realizar o compartilhamento destes recursos. Nuvens públicas são as mais utilizadas, que atendem as demandas dos clientes com uma cobrança sob demanda e de acordo com um contrato de prestação de serviços (*SLA - Service Level Agreement*). E as nuvens híbridas, que representam o modelo que mais tem crescido entre os consumidores dos serviços de computação em nuvem (CHANDRASEKARAN, 2014).

A adoção da computação em nuvem traz consigo uma série de benefícios e novas possibilidades. Agilidade para atender as demandas organizacionais relacionadas a necessidade de aumento da capacidade computacional, sem que isso impacte em altos custos com investimentos e longos prazos para aquisição de ativos. Devido a sua característica de *self-service*, as empresas podem poupar tempo e dinheiro, ao adaptar seus recursos computacionais disponíveis sem a necessidade de um complexo processo administrativo.

Outro ponto que tem relação direta com a economia financeira deste modelo de serviço é no ganho de escala, onde os provedores públicos de computação em nuvem conseguem ratear seus custos fixos com o volume de clientes. Além disso, as empresas podem experimentar novas aplicações, ferramentas e desenvolvimento de soluções sem antes realizar investimentos em *hardware*, *software* e infraestrutura de rede (HURWITZ et al., 2009).

A dificuldade econômica e apelo sobre fatores climáticos da atualidade irão potencializar a adoção deste modelo, principalmente por empresas sem muitos recursos para tais investimentos em suas estruturas próprias (LEAVITT, 2009). Além disso, problemas relacionados ao consumo energético, espaço e custos com tecnologia da informação afetam cada vez mais empresas. A evolução das soluções em nuvem facilitam cada vez mais a

migração de serviços para modelos de fornecimento público do serviço, inclusive, entre diferentes fornecedores.

Com a crescente demanda por fornecedores e soluções de computação em nuvem, alguns temas começam a ser debatidos, como a capacidade de um determinado fornecedor atender a todos os requisitos dos clientes, mantendo a credibilidade da sua prestação de serviços. Estes fornecedores por mais que tenham sob seu domínio um volume imenso de recursos de tecnologia, eles são finitos, seja pela capacidade de sua infraestrutura ou por limitações de software. Outros pontos de limitação podem estar relacionados a soluções específicas, em que o cliente pode se deparar com uma necessidade atendida por fornecedores especializados para atender a tal demanda, como é o caso do *Google* com suas soluções de mapas e georreferenciamento. Para estes cenários, construir um ambiente *multi-cloud* operando com mais de um fornecedor garante o acesso ao que cada um tem de melhor em seus ecossistemas, além de minimizar os riscos de perda de dados e tempo de inatividade decorrente da instabilidade das plataformas.

2.2 Multi-Cloud

A adoção de soluções em nuvem, por mais que seja tendência, ainda possui alguns desafios. Ser dependente de um único fornecedor deste tipo de serviço é a maior barreira na adoção de soluções de computação em nuvem, dependência essa que muitas vezes ocorre por falta de uma solução técnica aderente ao ecossistema tecnológico ou conhecimento técnico que permita o contrário (OPARA-MARTINS; SAHANDI; TIAN, 2016).

Implementar um ambiente *multi-cloud* significa que o escopo da solução de software não está restrito e limitado aos recursos oferecidos por um único fornecedor, mas que haja a possibilidade de utilização de cada um destes recursos em diferentes fornecedores, inclusive, a fragmentação de suas funcionalidades para que estas também possam ser distribuídas entre tais fornecedores.

Em geral, as razões que levam a escolha de um ambiente *multi-cloud* (KLAFFENBACH; KLEIN; SUNDARESAN, 2019):

1. Redundância para que não haja uma dependência única de fornecedor, e em caso de interrupção ou oscilação na prestação do serviço, leve à indisponibilidade do sistemas
2. Solução não atende as necessidades do cliente, sejam elas relacionadas à localização geográfica ou desempenho esperado
3. Fornecedor não oferece o serviço desejado, uma vez que cada um deles tendem a implementar soluções para necessidades específicas, aprofundando suas especialidades em determinadas áreas

2.2.1 Oferta de produtos do fornecedor

Um ecossistema *multi-cloud* garante às empresas a flexibilidade na escolha de soluções que possuem mais aderência às necessidades de suas áreas, como a facilidade da administração de recursos e suas funcionalidades, execução concorrente de cargas de trabalho, desempenho durante a extração de dados, gestão de perfis e acesso de usuários, segurança e governança de dados, leis locais quanto a privacidade de dados e consumo de soluções de tecnologia especializadas, como aprendizado de máquina, inteligência artificial, *big data*, dados geográficos, dentre outros.

Os grandes fornecedores de serviços de *cloud*, e que lideram globalmente a oferta de mercado são a *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)*, *Microsoft Azure* e a *Oracle Cloud Infrastructure (OCI)*, que possuem em seu portfólio de soluções serviços nas mais diversas áreas da tecnologia, como armazenamento, computação, compartilhamento, aplicativos, e muitos outros. Estes serviços possuem particularidades em suas respectivas plataformas, como uma parametrização específica, flexibilidade no consumo dos recursos, variação de desempenho e custos operacionais. Desta forma, a adoção de uma estratégia *multi-cloud* permite que as empresas façam uma escolha de tecnologias e soluções tirando o melhor proveito de cada um dos fornecedores.

Ao contratar soluções na modalidade *IaaS* ou *PaaS* é fundamental que haja alta disponibilidade da prestação de serviços, e em caso de falha pelo fornecedor, seja possível realizar a migração dos serviços para outro. Isto nem sempre pode ser garantido com soluções *SaaS*, por se tratarem de ofertas de serviços realizadas de forma específica por cada fornecedor.

Os benefícios de um ecossistema como este são, a resiliência a eventos de desastre e falhas do fornecimento do serviço, alta disponibilidade no acesso aos serviços oferecidos, flexibilidade operacional em aspectos financeiros, balanceamento de cargas de trabalho, manutenção ou por razões de conveniência, competitividade de preços entre fornecedores e diversidade na escolha de *hardware* conforme a demanda existente (RAJ; RAMAN, 2018).

Pontos negativos estão relacionados a aspectos referente aos diferentes recursos e modelos de cobrança adotados pelos fornecedores deste tipo de serviço, e que podem significar diretamente na quantidade de esforço necessário durante o desenvolvimento de aplicações em um ambiente *multi-cloud* (VARGHESE; BUYYA, 2017).

2.2.2 Restrições de escopo do cliente

A experiência no uso de aplicações em nuvem, as quais possuem usuários distribuídos globalmente, sofre grande impacto sobre qual estratégia foi adotada na contratação de nuvens computacionais e suas respectivas localizações geográficas, utilizar múltiplos fornecedores para garantir que as requisições sejam respondidas mais rapidamente. Outro

ponto que precisa ser levado em consideração são os requisitos legais, alguns países restringem ou regulamentam a transferência de dados entre servidores a fim de proteger dados pessoais de sua população.

Existem ainda cenários referente às obrigações relacionadas ao armazenamento e processamento de informações em estruturas de computação em nuvem, que possuem especificações de segurança regionais e que devem ser compreendidas pelos fornecedores deste tipo de serviço (BOHLI et al., 2013). Alguns exemplos que podem ser citados neste escopo:

- O processamento de informações médicas de moradores dos Estados Unidos requer a certificação HIPAA (*Health Insurance Portability and Accountability Act*);
- O processamento de informações utilizando dados de cartão de crédito requer a certificação PCI DSS (*Payment Card Industry Data Security Standard*);
- Empresas estabelecidas na União Europeia só podem contratar fornecedores de fora da denominada área econômica europeia se aderirem ao GDPR (*EU Data Protection Directive*);
- No Brasil, a LGPD (Lei Geral de Proteção de Dados) define que qualquer operação de tratamento de dados ofertados no território brasileiro estão suscetíveis às sua regras e políticas de segurança.

Cenários em que restrições de escopo do cliente sejam aplicáveis demandam flexibilização no escalonamento de aplicações e suas instâncias em ambientes *multi-cloud*. Como por exemplo, requisitos regulatórios, qualidade na experiência de uso e latência, e assim, demandando um modelo flexível de distribuição de aplicações *web*, em que o balanceamento da requisição do usuário acontece de acordo com sua localização geográfica, obtida a partir de seu endereço IP (GROZEV; BUYYA, 2016).

2.2.3 Arquiteturas *Multi-Cloud*

No trabalho de (HOHPE, 2020), o autor faz uma completa análise do processo de migração de infraestruturas computacionais privadas para ecossistemas de computação *multi-cloud*, aproveitando ao máximo suas facilidades, como a grande variedade na oferta de produtos e serviços, rápida expansão de recursos computacionais, auto otimização, autocorreção a erros de *hardware* e *software*, tarifação pelo consumo e distribuição global. A estratégia de migração é algo complexo, não existindo um padrão para aplicá-lo em larga escala, cabendo a cada organização realizar a análise de seus requisitos técnicos e restrições.

Ao classificar as arquiteturas *multi-cloud*, é possível organizá-las nos seguintes cenários de uso deste tipo de ambiente:

- **Arbitrário**: As cargas de trabalho pode estar disponíveis em mais de uma nuvem, mas sem um motivo específico;
- **Segmentado**: Diferentes nuvens são utilizadas para diferentes objetivos;
- **Escolha**: Projetos, aplicações ou unidades corporativas optam por um fornecedor específico;
- **Paralelo**: Uma única aplicação é distribuída de forma simultânea em mais de uma nuvem computacional, com objetivo de garantir altos índices de disponibilidade do serviço;
- **Portável**: As cargas de trabalho podem ser migradas facilmente entre os fornecedores conforme a necessidade.

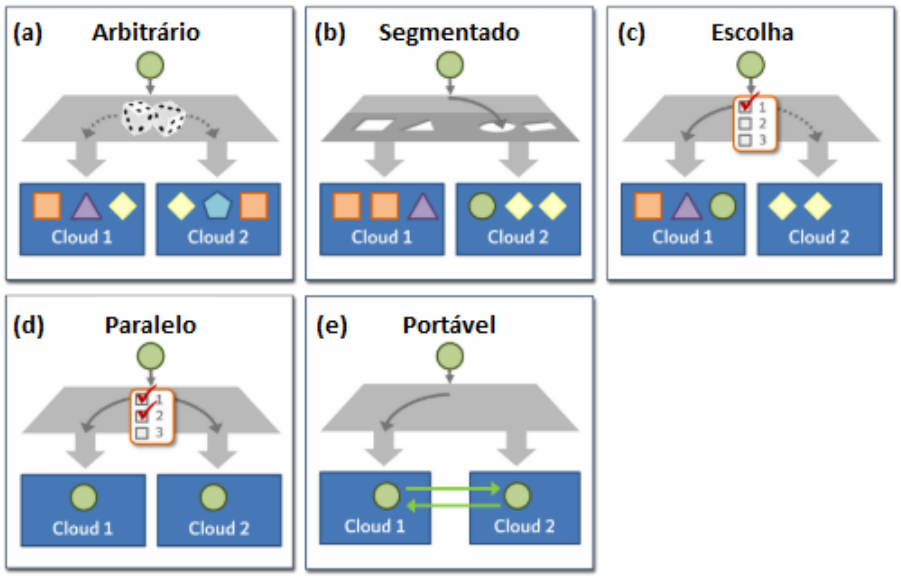

Figura 1 – Cenários de Arquitetura *Multi-Cloud*. Adaptada de (HOHPE, 2020)

No cenário de *multi-cloud* arbitrária (Figura 1 (a)), todas as cargas de trabalho estarão distribuídas entre múltiplos fornecedores sem um motivo aparente. Esta escolha pode não possuir uma definição clara pelo motivo ao qual foram direcionadas a um determinado fornecedor, mas geralmente sofrem influência em possíveis concessões de créditos a título de cortesia para degustação da plataforma, afinidade da equipe técnica, seja pela tecnologia disponível para uso ou preferência por uma determinada marca. Por não haver uma estratégia bem definida no processo de escolha, isto não é uma decisão totalmente errada, afinal alguns serviços estão sendo migrados para a computação em

nuvem, ao mesmo tempo que se ganha experiência e amadurecimento, tanto por parte da organização quanto pela equipe técnica.

Por sua vez, o cenário de *multi-cloud* segmentado (Figura 1 (b)) garante uma boa prática de planejamento do ecossistema. Nele a distribuição das aplicações ocorre dentro de fornecedores de acordo com sua especialidade e oferta de serviços, desta forma, a melhor prestação de serviço de cada fornecedor pode ser extraída.

Critérios para a segmentação das cargas de trabalho podem levar em consideração as suas características (compatibilidade com tecnologias utilizadas e atendimento a dependências requeridas, como sistema operacional ou arquitetura de hardware), nível de confidencialidade de dados requerida e a categoria a qual o produto se enquadra, como máquinas virtuais com finalidade para computação geral, análise de dados, inteligência artificial, dentre outros.

Alguns fatores impactam na construção de tais critérios, como a rápida evolução das competências de cada fornecedor, onde a cada dia têm expandindo seu leque de ofertas, fazendo com que estes critérios se tornem obsoletos. Além disso, estratégias comerciais em determinados períodos podem tornar satisfatórias a migração de um determinado serviço hospedado em um fornecedor para outro, forçando uma mudança do cenário segmentado para o arbitrário, ignorando o modelo de critérios pré-definido.

Já o cenário de *multi-cloud* definido por escolha de fornecedor (Figura 1 (c)), independente da carga de trabalho a ser distribuída, pode remeter o ecossistema a um verdadeiro ambiente *multi-cloud*, minimizando a probabilidade de *lock-in* no fornecimento dos serviços. Nele é garantido a heterogeneidade das aplicações mediante a capacidade e limitações de cada um dos fornecedores, o que significa que a distribuição pode ser realizada livremente entre todos os disponíveis a qualquer momento.

A opção pelo cenário de escolha ainda pode ser considerada uma prática de governança da tecnologia de informação, envolvendo grandes empresas que compartilham serviços de tecnologia em vários níveis. Uma divisão responsável, centralizadora das áreas de tecnologia, homologam fornecedores para plataformas de computação em nuvem, enquanto as áreas subordinadas optam pelas soluções que mais se aderem às suas estratégias e conhecimento técnico da equipe. Isto facilita a administração financeira e monitoramento dos gastos.

No cenário paralelo (Figura 1 (d)), aplicações consideradas críticas podem ser disponibilizadas de forma simultânea através de vários fornecedores, garantindo assim altos níveis de disponibilidade em suas aplicações. Entretanto, um aumento do custo é inevitável, pois uma mesma carga de trabalho será disponibilizada em várias nuvens computacionais simultaneamente.

O cenário portável (Figura 1 (e)) tem o objetivo de garantir a liberdade de migração das cargas computacionais entre todos os fornecedores disponíveis, garantindo que não haja *lock-in* com algum fornecedor, e ainda, conceder poder de negociação para as empresas para buscar as melhores ofertas de nuvem pública no mercado. Além disso, a movimentação das aplicações entre o ecossistema *multi-cloud* pode ser baseada em necessidades personalizadas, como um monitoramento do consumo da aplicação ou localização geográfica dos usuários.

Implementar este cenário requer a adoção de ferramentas capazes de realizar a orquestração, além disso, existe a complexidade de configuração deste ambiente, bem como um *lock-in* na ferramenta escolhida.

A Tabela 1, faz um resumo dos possíveis cenários de operação em uma arquitetura *multi-cloud*, apresentando suas características principais e pontos de atenção a serem levados em consideração.

Cenário	**Atributo Chave**	**Mecanismo Chave**	**Considerações**
Arbitrário	Distribuir aplicativos em nuvem	Habilidade em nuvem	Perda de governança e custos extras com tráfego de rede
Segmentado	Objetivo claro sobre o uso da nuvem	Governança	Similar ao cenário arbitrário
Escolha	Projetos com necessidades específicas e redução do *lock-in*	Solução comum para provisionamento, faturamento e governança	Camada extra de abstração
Paralelo	Alta disponibilidade	Automação, abstração e balanceamento de carga	Complexidade e subutilização dos serviços em nuvem
Portável	Balanceamento das cargas de trabalho conforme necessário	Automação completa, abstração e portabilidade dos dados	Complexidade, *lock-in* em ferramentas *multi-cloud* e subutilização

Tabela 1 – Resumo das diferentes arquiteturas em uma operação *multi-cloud*. Adaptada de (HOHPE, 2020)

2.3 Virtualização e Contêineres

Virtualização pode ser definida como uma tecnologia com um papel essencial em infraestruturas computacionais em nuvem. Um método utilizado para consolidar diferentes tipos de recursos e possibilitar múltiplos usuários acessá-los através de sua representação virtual, criando uma versão dos recursos como memória, processamento, armazenamento, etc., acessível através de uma nova camada (NGUYEN, 2019).

Uma tecnologia que combina ou divide os recursos computacionais em um ou mais ambientes, utilizando metodologias para realizar o particionamento ou agregação

de *hardware* e *software*, de forma parcial ou completa para tarefas como simulação de máquinas, emulação, compartilhamento e muitos outros (NANDA; CHIUEH, 2005).

Tecnologias para virtualização computacional fornecem um ecossistema que simula uma máquina dedicada disponível para execução das cargas de trabalho, garantindo o isolamento entre tais máquinas, inclusive, com o próprio ambiente hospedeiro. Basicamente, máquinas virtuais são executadas sob um ambiente chamado *hypervisor*.

O *hypervisor* é um programa que controla e gerencia o sistema operacional, sendo esta uma das tecnologias mais utilizadas e difundidas para realização desta tarefa, aliada principalmente ao fato de sua simplicidade e características de fornecer uma plataforma aberta para implementação de seus recursos. Basicamente este programa permite a execução de uma ou mais máquinas virtuais sob um mesmo *hardware* hospedeiro (DHAR, 2016). Algumas plataformas comerciais que utilizam a tecnologia de *hypervisor* são *Xen*[1], *Virtual box*[2], *VMWare*[3], *Parallels*[4], etc.

No contexto das máquinas virtuais, cada instância virtual em execução é denominada como sendo um sistema convidado (*guest OS*), enquanto que o sistema principal em execução no hardware é chamado de sistema hospedeiro (*host OS*). Algumas plataformas comerciais de mercado que utilizam esta tecnologia são: *Xen*, *Virtual Box* e *VMWare* (DHAR, 2016).

Os *hypervisors* existentes para construção de ambientes para máquinas virtuais podem ser organizados em duas categorias: tipo 1 e tipo 2 (PRASAD, 2014). Os de tipo 1 são executados sob a camada de hardware, também conhecido como máquinas virtuais nativas, uma vez que eles substituem o sistema operacional hospedeiro e realizam o acesso ao hardware diretamente, como ilustrado na Figura 2. Sendo executado em modo *kernel*, possuem acesso exclusivo ao CPU físico, inclusive, garante que se uma máquina virtual hóspede falhar por qualquer motivo, nenhuma outra máquina virtual será afetada.

Já os *hypervisors* tipo 2 são aplicações instaladas sob o sistema operacional, assim como qualquer outra aplicação, neste caso, o sistema operacional hospedeiro é responsável pela alocação de recursos computacionais, tornando assim uma arquitetura de três camadas, conforme representado na Figura 3.

Alguns dos benefícios em se aderir a estratégias de virtualização computacional, está na flexibilidade em alocar recursos de hardware disponíveis de acordo com as demandas dos ambientes nele implementados, além de permitir a escalabilidade de aplicações e a fácil replicação dos ambientes já configurados. Em uma visão administrativa, o compartilhamento de um único ambiente físico em múltiplos ambientes virtualizados garante uma redução

[1] <https://xenproject.org>
[2] <https://virtualbox.org>
[3] <https://vmware.com>
[4] <https://parallels.com>

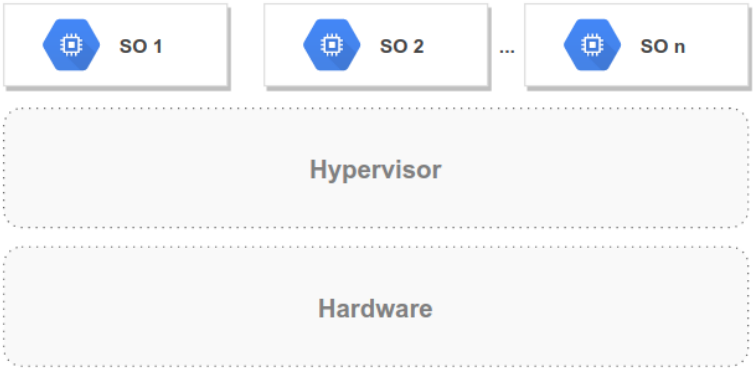

Figura 2 – Arquitetura de Máquinas Virtuais de categoria "Tipo 1". Adaptada de (PRASAD, 2014)

Figura 3 – Arquitetura de Máquinas Virtuais de categoria "Tipo 2". Adaptada de (PRASAD, 2014)

dos custos operacionais que envolvem o fornecimento de soluções para processamento em nuvem (HWANG; DONGARRA; FOX, 2013).

Em relação à alta disponibilidade durante o uso de ambientes virtualizados, esta abordagem garante que se uma das instâncias virtualizadas sofra alguma interrupção, não afetará as demais em execução, isto porque nos ecossistemas de servidores virtualizados, cada instância computacional está alocada em diferentes máquinas virtuais e com seu gerenciamento isolado. Possibilitando inclusive que múltiplos sistemas operacionais sejam utilizados sob um mesmo *hardware* (TANENBAUM; BOS, 2014).

Outros benefícios da virtualização computacional vão além da economia financeira com a aquisição de equipamentos de *hardware*, consumo de energia e espaço para armazenamento físico, elas apoiam a implementação de novas soluções sem a necessidade de investimento com a aquisição de equipamentos a cada nova demanda. Além disso, permite que cada aplicação possa utilizar seu próprio ecossistema de sistema operacional,

bibliotecas e parâmetros, já que cada uma delas terá seu próprio ambiente para execução. Este isolamento de ambiente também é muito útil para a execução de aplicações legadas, que por seus próprios motivos precisam ser implementadas em versões específicas de sistemas operacionais (TANENBAUM; BOS, 2014).

O conceito por trás dos contêineres é similar às máquinas virtuais, porém mais eficiente em relação ao seu tempo de inicialização e consumo de recursos. Projetados inicialmente para satisfazer demandas de serviços em nuvem, um contêiner encapsula todos os pacotes e dependências necessárias para execução da aplicação, simplificando seu uso e manipulação através de modelos voltados a orquestração da distribuição de aplicações através de nuvem públicas e privadas (YADAV; GARG; MEHRA, 2019).

Seu mecanismo principal responsável por iniciar e interromper a sua execução é comparado ao *hypervisor* utilizado pelas máquinas virtuais, a diferença está no gerenciamento dos processos em execução, que nos contêineres todos são equivalentes a processos nativos no sistema operacional hospedeiro, já nas máquinas virtuais estão ligados ao processo principal do *hypervisor* (MOUAT, 2015).

Por mais que haja convergência entre os objetivos no uso e aplicação de máquinas virtuais e contêineres, o conceito utilizado por estas tecnologias são diferentes. Uma máquina virtual emula um ambiente computacional completo, com a configuração de todos os componentes de *software* necessários, já os contêineres tornam as aplicações portáveis e auto contidas em relação às suas dependências e requisitos para sua execução (MOUAT, 2015). Contêineres são aplicações encapsuladas com suas dependências para execução de forma independente.

Durante a construção de ambientes virtualizados utilizando contêineres, ao considerarmos diferentes aplicações que compartilham as mesmas bibliotecas e dependências, haverá um reaproveitamento destes recursos, sem a necessidade de cópias redundantes em cada um dos ambientes. Todo este reaproveitamento se traduz em contêineres que utilizam pouco espaço em disco e que necessitam de poucos segundos para sua inicialização completa, com isso, o trabalho de desenvolvedores e administradores de sistema fica menos complexo (MERKEL, 2014).

Os conceitos fundamentais na implementação da tecnologia de contêineres são os mesmos daqueles utilizados em máquinas virtuais, as diferenças entre tais tecnologias estão relacionadas basicamente na camada lógica em que são executadas - diretamente sob o *hardware* através do *hypervisor* ou sob um sistema operacional hospedeiro com o compartilhamento de seu *kernel* - e nos recursos necessários para sua execução. Compartilhando o mesmo *kernel* entre o contêiner e o sistema hospedeiro, garante a esta tecnologia a característica de ser mais “leve” e compacta, se comparada às máquinas virtuais tradicionais, além disso, a interoperabilidade durante a execução das aplicações (YADAV; GARG; MEHRA, 2019).

Um ecossistema baseado em contêineres fornece acesso compartilhado ao sistema operacional hospedeiro, incluindo um acesso único ao sistema de arquivos, executáveis do sistema e bibliotecas, além disso, durante a definição dos aspectos de um contêiner é possível definir outros recursos que estarão disponíveis, como bibliotecas e *software* de terceiros. Cada contêiner pode ser ligado, desligado e reiniciado como um sistema completo, porém em apenas uma fração de seu tempo, e para as aplicações e usuários dele, é como se fosse um sistema exclusivo (SOLTESZ et al., 2007).

O uso de contêineres para execução de aplicações tem como principal pilar o conceito de seu isolamento no nível de sistema operacional, possibilitando assim a execução de múltiplas aplicações em um mesmo ambiente sem que haja interferência entre elas, isto permite, por exemplo, a execução de aplicações em contêineres utilizando um sistema operacional X, tendo como sistema operacional Y no hospedeiro. Tal isolamento é possível através da utilização de recursos existentes no *kernel* do sistema operacional Linux, os *namespaces* e *control groups*, sendo o *namespaces* responsável por permitir que os diferentes componentes do sistema operacional sejam acessados e utilizados em *workspaces* - neste cenário, tais *workspaces* são representados pelos contêineres - enquanto os *control groups* permitem um controle rigoroso sobre a utilização dos recursos de hardware do sistema hospedeiro, de tal forma que não seja possível que um contêiner consuma todos os recursos disponíveis no sistema (SCHOLL; SWANSON; JAUSOVEC, 2019).

Diferente de máquinas virtuais, os contêineres podem compartilhar dados com o sistema operacional hospedeiro - o que garante uma execução leve no tocante a consumo de memória e disco para armazenamento -, ficando este por sua vez responsável por gerenciar o acesso compartilhado a tais dados. Neste modelo de virtualização não há acesso direto ao *hardware*, mas sim através do sistema operacional hospedeiro (EDER, 2016).

Conforme representado na Figura 4, é possível observar as camadas de isolamento dos ecossistemas virtualizados, e como acontece compartilhado ao *kernel* do sistema operacional hospedeiro no modelo utilizando contêineres.

Por conta disso, o uso de contêineres tem atraído cada vez mais a atenção de quem precisa de uma ferramenta para implementações de funcionalidades que demandam um alto nível de escalabilidade. A Figura 5 demonstra a diferença dos cenários utilizando contêineres e máquinas virtuais em execução no mesmo sistema hospedeiro, destacando a abstração e compartilhamento no uso de recursos.

A estrutura de compartilhamento de *kernel* do sistema operacional utilizada pelos contêineres permite isolamento suficiente para garantir a segurança de execução de cada um deles, de tal forma que não haja comprometimento ou um acesso não autorizado, suscetível apenas à falhas no sistema operacional, sendo esta a camada comum de compartilhamento entre eles, diferente de ambientes virtualizados, que possuem um acoplamento total do ecossistema necessário para sua execução. Os principais pontos em que a estrutura de

Figura 4 – Arquitetura de máquinas virtuais em *hypervisor* versus contêineres. Adaptada de (YADAV; GARG; MEHRA, 2019)

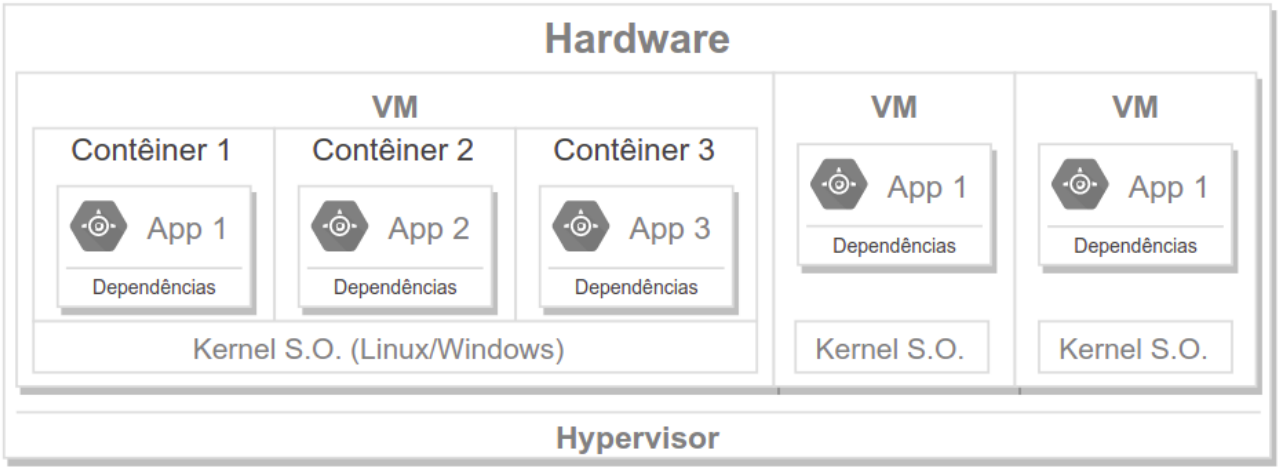

Figura 5 – VMs e contêineres em execução num mesmo hospedeiro. Retirada de (SCHOLL; SWANSON; JAUSOVEC, 2019)

contêineres se sobressai frente a máquinas virtuais (SCHOLL; SWANSON; JAUSOVEC, 2019):

- Máquinas virtuais levam um tempo consideravelmente maior para iniciar, pois carrega um ambiente de sistema operacional completo;
- O tamanho das máquinas virtuais pode ser um problema, pois como contém um sistema operacional completo, podem possuir facilmente vários GB (*gigabytes*) de tamanho, o que torna um problema o processo de replicação entre diferentes *data-centers*.

- Realizar o escalonamento de máquinas virtuais possui alguns desafios. Escalonar novas cargas de trabalho requer a construção de novas instâncias, aumentar recursos de processador, memória, armazenamento, etc. requer a reinicialização do sistema como um todo. Desta forma, escalonar uma aplicação pode não acontecer na velocidade em que a demanda necessite de tal recurso.

- Máquinas virtuais requerem um maior poder computacional para sua execução, como processador, memória, armazenamento, etc. limitando assim a quantidade possível para execução em um único ambiente hospedeiro.

Em ambientes baseados em contêineres, o conceito fundamental na distribuição de aplicações está em garantir que a implementação realizada no ambiente de desenvolvimento, sob um conjunto definido de ferramentas e bibliotecas, será executada como esperado no ambiente onde será hospedado, sendo este um *datacenter* local, em uma nuvem computacional - pública e/ou privada - ou no computador de um usuário, independente inclusive do sistema operacional hospedeiro ou da arquitetura de hardware. Profissionais arquitetos de ambientes computacionais podem desta forma, se preocupar com características voltadas a segurança no acesso à dados e disponibilização geral no acesso a aplicação, sem se preocupar com dependências de software, conflitos ou restrição de versões (KLAFFENBACH; KLEIN; SUNDARESAN, 2019).

2.3.1 *Docker* e *Kubernetes*

Disponibilizada inicialmente em 2013, como uma ferramenta de código aberto com foco em solução para construção, execução e distribuição de aplicações conteinerizadas, junto com suas ferramentas dependentes, dentro em um ecossistema virtualizado (PALAN, 2018), a plataforma *Docker* trouxe um conjunto de funcionalidades para auxiliar a adoção e implementação de ecossistemas utilizando a tecnologia de contêineres implementada no *kernel* o sistema operacional *linux*, o expandindo de algumas formas, inicialmente facilitando o conceito de criação de imagens portáteis - de fácil compartilhamento, distribuição e replicação - e fornecendo uma interface de utilização amigável (MOUAT, 2015).

A plataforma *Docker* oferece um ambiente para gerenciamento de imagens de contêineres e uma biblioteca para sua execução, sendo responsável pelo seu ciclo de vida, suporte a gestão utilizando linhas de comandos e ainda *API* para gerenciamento do ecossistema (ARUNDEL; DOMINGUS, 2019).

Seu ecossistema de funcionamento pode ser separado em dois componentes distintos, seu mecanismo principal, responsável pela criação e execução dos contêineres e pela plataforma *Docker HUB*, um serviço em nuvem para distribuição e compartilhamento de contêineres.

Os benefícios da utilização do *Docker* está no fornecimento de um ambiente rápido e conveniente para execução de contêineres, onde antes disso, a execução de um contêiner utilizando tecnologia *LXC* necessitava conhecimento avançado e muito trabalho manual na parametrização do ambiente. Já o *Docker HUB* provê um ambiente com inúmeras imagens de contêineres compartilhadas, agilizando a construção e configurações de ambientes ao utilizar imagens já disponibilizadas por outros usuários (MOUAT, 2015).

Ao realizar a migração de múltiplas aplicações para um ecossistema baseado em contêineres aumenta-se a complexidade na gestão de todas as cargas de trabalho em execução. A tarefa de orquestração garante que todos sejam executados e estejam disponíveis conforme foram configurados.

A popularização do modelo computacional utilizando contêineres fez com que várias ferramentas surgissem com o propósito de realizar a tarefa de orquestração, fornecendo mecanismos para uma estratégia eficiente na manutenibilidade de todos os contêineres disponíveis, tais como: recuperação automática daqueles que geraram algum tipo de falha na execução, gerenciamento de atualizações de versões com o mínimo tempo de indisponibilidade, ou ainda, garantir um crescimento elástico de acordo com o aumento da demanda, com a disponibilização de instâncias extras para suprir o crescimento da demanda (BAIER; SAYFAN; WHITE, 2019).

Dentre as opções disponíveis para utilização, a ferramenta mais conhecida é o *Kubernetes* - também conhecido como *k8s* - projeto *open source* iniciado pela *Google* em 2014, responsável por realizar a execução, gerenciamento e portabilidade de contêineres entre diferentes nuvens computacionais. A funcionalidade principal do *Kubernetes* pode ser definida como uma ferramenta orquestradora de contêineres, sendo ele responsável pela alta disponibilidade dos artefatos disponibilizados em sua estrutura, com destaque ao monitoramento com substituição das instâncias que por algum motivo fiquem inacessíveis, seja por falha de rede, encerramento inesperado, ou qualquer outro evento que as remetem a um estado de inacessibilidade (SAYFAN, 2018).

A Figura 6 ilustra a arquitetura da ferramenta *Kubernetes*, a qual realiza a gestão de diferentes nós computacionais, e conforme a demanda de implantação de novas instâncias da aplicação, realiza o provisionamento dos recursos necessários de forma automática.

O componente *master* na arquitetura do *Kubernetes* é responsável pelos processos de segurança que envolvem autorização de credenciais e autenticação, fornecimento de *API* para integração com outras ferramentas, construção e disponibilização de contêineres, escalonamento, replicação e gerenciamento do *cluster* computacional, o componente *node* é subordinado ao *master* e é responsável por realizar a execução propriamente dita dos contêineres *Docker*. Já o componente *pod* representa as unidades de trabalho de um ou mais contêineres, sendo ele, a menor unidade computacional possível. Cada *pod* possui isolamento de informações de acordo com as premissas do *kernel* do sistema operacional

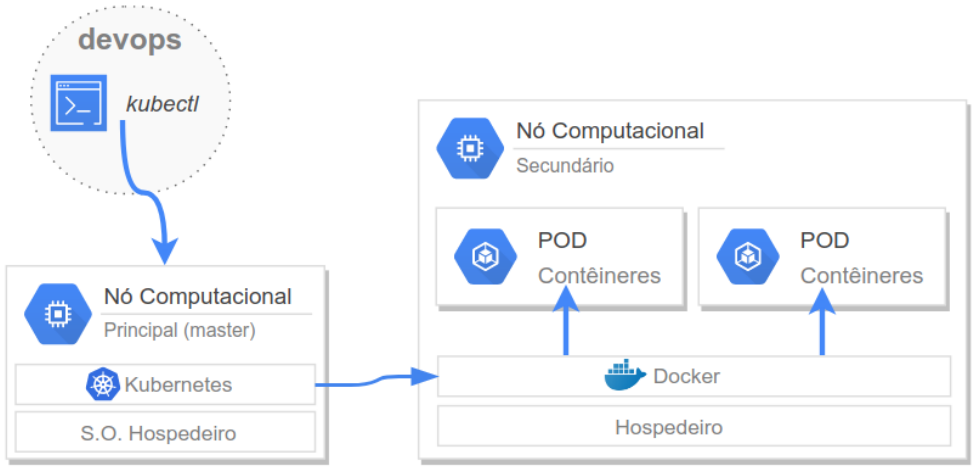

Figura 6 – Arquitetura *Kubernetes*. Adaptada de (RAJ; RAMAN, 2018)

Linux, como os identificados de processos (*PID*), configurações de rede, comunicação entre os processos (*IPC*) e relógio compartilhado (*UTS*).

O *Kubernetes* entrega uma suíte de ferramentas aos seus utilizadores para simplificar processos de disponibilização de cargas de trabalho dentro de uma infraestrutura disponível para tal. Esta e outras ferramentas garantem que o processo como um todo seja simples, absorvendo toda a complexidade inerente. Além disso, tais características não atuam apenas como uma plataforma de gerenciamento de sistemas, mas como um norteador de boas práticas na implementação e distribuição de sistemas computacionais em nuvem, utilizando padrões de projeto de alto nível (SAYFAN, 2018).

Realizar o escalonamento das cargas de trabalho é uma tarefa fundamental das ferramentas de orquestração, envolvendo para isso o monitoramento de forma constante do ambiente em execução em atenção às mudanças. No Capítulo 3 serão abordadas as estratégias de escalonamento comumente utilizadas, suas características e processos envolvidos.

3 Orquestração de Contêineres

A orquestração de contêineres consiste em um conjunto de tarefas que compreende a seleção, implantação, monitoramento e controle dinâmico das cargas de trabalho em um *cluster* computacional, para isso, respeitando as configurações de cada contêiner. Esta orquestração permite o gerenciamento das aplicações de forma automática e controle dinâmico de ambientes que possuem vários contêineres disponibilizados em nuvem (TOSATTO; RUIU; ATTANASIO, 2015; CASALICCHIO, 2017).

Os orquestradores implementam um modelo orientado a serviços, onde os consumidores se preocupam na definição dos requisitos da aplicação, a partir dos recursos existentes nos ecossistemas dos fornecedores de nuvens computacionais. E assim, o objetivo do orquestrador é garantir a implantação e distribuição das aplicações em conformidade com o que foi definido (WEERASIRI et al., 2017).

Os processos que envolvem a orquestração de contêineres podem ser organizados entre: **porquê**, **quando** e **onde** uma aplicação irá disparar eventos ao orquestrador. Tais eventos podem ser referentes a uma nova execução, migração ou desativação de uma instância da aplicação. O **porque** pode ocorrer quando há necessidade de aumento dos recursos computacionais devido ao uso intenso de um serviço, ou ainda, em situações de falhas em um dos nós, através de uma verificação realizada pelo orquestrador. A decisão pelo **quando** e **onde** uma instância do contêiner será disponibilizada leva em consideração as configurações específicas durante a sua criação, como a definição de regras restritivas ou afinidade entre os contêineres (CASALICCHIO, 2017).

Um ecossistema computacional orquestrado está em um constante processo de monitoramento das cargas de trabalho em execução, dos recursos computacionais disponíveis e na demanda atual no acesso aos serviços disponibilizados. O processo de orquestração de uma aplicação se inicia no mapeamento dos seus parâmetros e requisitos para execução, após isso, a infraestrutura deve ser preparada. Esta etapa envolve a definição dos recursos de *hardware* necessários para execução de uma nova instância da aplicação, bem como a disponibilidade destes recursos dentro do *cluster* computacional, a fim de se ter a definição do local de disponibilização da mesma. O próximo passo será a preparação do ambiente hospedeiro, que inclui o *download* das imagens do contêiner e binários de ferramentas dependentes. Após a disponibilização da nova instância, inicia-se o seu monitoramento, realizando a coleta de métricas para o orquestrador identificar se a nova alocação foi suficiente para suprir a demanda (ANTONESCU; ROBINSON; BRAUN, 2012).

Uma ferramenta de orquestração fornece mecanismos para implementar funcionalidades na gestão da disponibilização de contêineres, como a limitação no consumo de

recursos, escalonamento, balanceamento de carga, verificação de disponibilidade, tolerância a falhas e elasticidade. (SCHOLL; SWANSON; JAUSOVEC, 2019) classificam as principais tarefas de um orquestrador como sendo:

- Realizar o provisionamento e construção dos contêineres nos recursos computacionais disponíveis;
- Gerenciamento constante dos contêineres em execução, realizando a sua disponibilização em nós computacionais com recursos suficientes e a migração de contêineres entre nós computacionais de acordo com o aumento ou diminuição da demanda;
- Monitoramento da "saúde" dos contêineres, realizando tarefas de inicialização e reinicialização em casos de falhas no contêiner ou a nível de nó computacional;
- Escalonamento dos contêineres no *cluster* computacional;
- Mapeamento e regras de acesso à rede;
- Balanceamento de carga entre os contêineres.

Estas funcionalidades apoiam a gestão de múltiplos ambientes computacionais, fazendo com que vários provedores de nuvem sejam administrados como se fossem um único. Além disso, garantem que a configuração de todas as dependências, em todas as nuvens computacionais, estejam idênticas e compatíveis para disponibilização das cargas de trabalho (RAJ; RAMAN, 2018). As principais funcionalidades realizadas por um orquestrador são:

1. Controlar os recursos disponíveis para limitar a interferência entre os contêineres, já que um contêiner pode ser capaz de utilizar todo o recurso disponível do ecossistema hospedeiro. Esta funcionalidade se torna fundamental durante o processo decisório de escalonamento de novas instâncias da aplicação (CASALICCHIO, 2019).
2. O balanceamento de carga é responsável por realizar a distribuição de novas instâncias da aplicação entre os nós do *cluster* computacional. Cada ferramenta de orquestração de contêineres possui algoritmos específicos que são utilizados durante esta tarefa. Por exemplo, o *Docker Swarm* possui três algoritmos, *spread* em que os novos contêineres são disponibilizados nos nós computacionais que possuem o menor número de contêineres em execução, *binpack* que considera o nó computacional com a maior quantidade de contêineres em execução, mas que possui ociosidade de recursos, e *round-robin*, forma aleatória tradicional. Nas ferramentas *Aurora* e *Marathon* esta tarefa é mais simples, de forma aleatória o nó computacional que possui recursos disponíveis é selecionado. Já no *Kubernetes*, em uma de suas implementações, é utilizado o algoritmo *best-fit*, através da política de priorização chamada

MostRequestedPriority, em que a escolha é realizada otimizando a quantidade de nós. Assim como no escalonador, durante as tarefas de balanceamento de carga também é possível realizar a personalização de regras, levando em consideração outras políticas (CASALICCHIO, 2019; TRUYEN et al., 2020).

3. Realizar o controle de disponibilidade dos contêineres, aplicado sob aqueles que estão em execução e sua capacidade na resolução de requisições HTTP, com isso, aferir o seu estado operacional. Esta configuração pode ser realizada em relação ao seu comportamento na resolução das requisições recebidas, como o prazo para *timeout*, a quantidade de verificações que serão realizadas ou ainda a quantidade de falhas consecutivas para considerar uma instância como inoperante. Diferentes estratégias de verificação ainda podem ser utilizadas de acordo com a ferramenta de orquestração utilizada. Por exemplo, a ferramenta *Docker Swarm* utilizada *socket* para verificação da situação do contêiner, já o *Kubernetes* possui serviços de monitoramento através de seu componente *kubelet agent*, que é disponibilizado junto com todos os contêineres, garantindo que estejam executando normalmente (TRUYEN et al., 2020).

4. Tolerância a falhas permite a implementação de regras relacionadas ao controle sob o número de réplicas ativas de uma determinada aplicação, e utiliza como apoio a verificação de disponibilidade do contêiner. Esta combinação de recursos garante ao orquestrador a capacidade de manter uma aplicação sempre disponível, e em caso de falha em alguma de suas instâncias, disponibilizar automaticamente uma nova (CASALICCHIO, 2019).

5. Por fim, a funcionalidade de elasticidade permite que instâncias de contêineres sejam disponibilizadas ou removidas conforme a demanda no consumo da aplicação, utilizando para isso políticas sob o consumo de recursos computacionais do ambiente hospedeiro, sendo possível personalizar tais regras, construindo um algoritmo que realize tal elasticidade a partir de políticas específicas (TRUYEN et al., 2020).

Durante a execução das cargas de trabalho o orquestrador monitora a necessidade no aumento ou diminuição da demanda computacional dos contêineres. Esta tarefa pode respeitar duas metodologias para suprir a elasticidade da carga computacional, a horizontal, que realiza a adição de novas instâncias do contêiner, realizando um balanceamento de carga entre elas, e a vertical, ajustando os recursos de CPU, memória e armazenamento de forma dinâmica.

A Figura 7 apresenta a arquitetura de um ambiente utilizando múltiplos nós computacionais, estejam eles hospedados em um mesmo *datacenter* ou em diferentes localidades, e que em cada um destes nós vários contêineres estão disponíveis. Neste ecossistema, o orquestrador se comunica com o mecanismo principal responsável pelo

gerenciamento dos contêineres em cada um dos nós e passa a coordenar a execução de cada um deles.

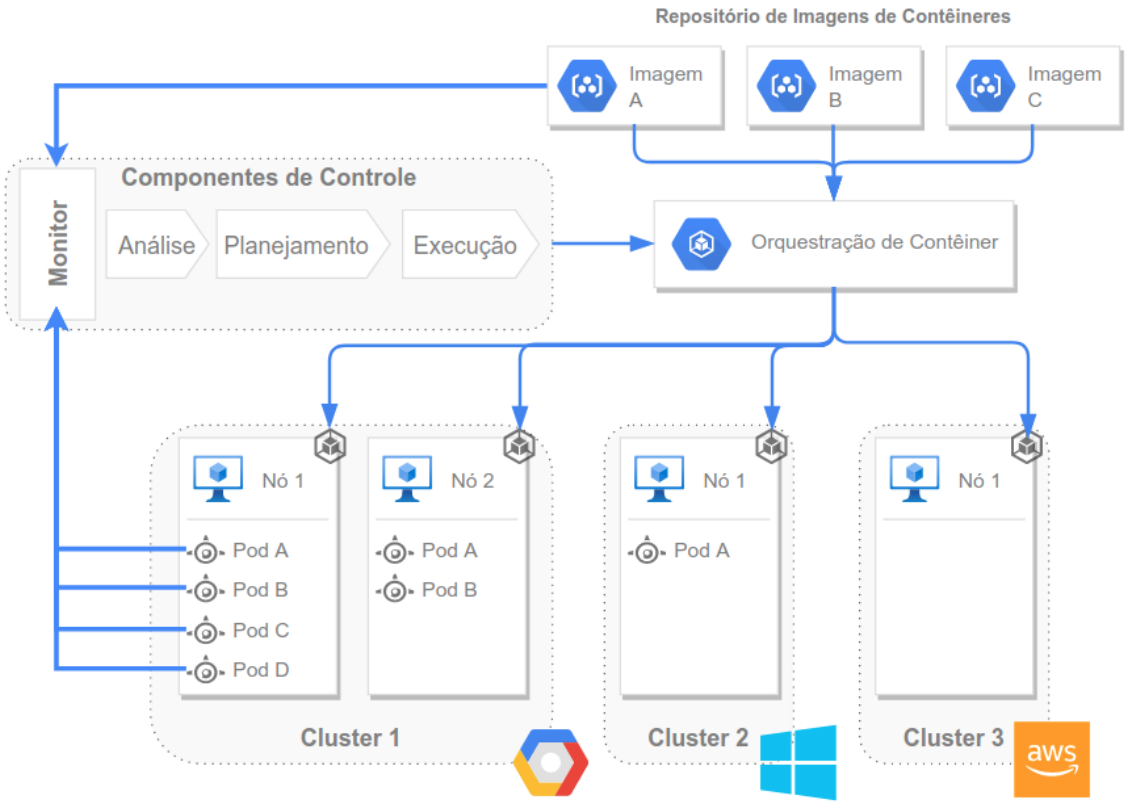

Figura 7 – Orquestração de contêineres. Adaptada de (CASALICCHIO, 2017)

3.1 Trabalhos Relacionados

Na literatura existem várias discussões e abordagens sobre a construção de alternativas aos algoritmos padrões utilizados pelas ferramentas de orquestração, envolvendo os processos de escalonamento, implantação e migração das cargas de trabalho. A grande maioria deles possui um foco relacionado à computação verde, favorecendo a aquisição de serviços de fornecedores que possuem *datacenters* que utilizem fontes de energias renováveis, foco na eficiência energética para redução dos custos operacionais e distribuição geográfica.

As principais preocupações estão relacionadas ao desempenho deste processo e a forma como ele acontecerá, uma vez que não envolvem intervenção humana e a velocidade na realização da tarefa de escolha sob qual nó computacional irá disponibilizar a aplicação devem estar de acordo com níveis mínimos no cumprimento de *SLAs*.

Os trabalhos apresentados a seguir foram considerados relevantes pela maneira a qual utilizam algoritmos construídos de forma personalizada para atender aos objetivos alvo de suas respectivas pesquisas.

(RODRIGUEZ; BUYYA, 2018) propõem um algoritmo para orquestração de contêineres com três objetivos. O primeiro deles é a otimização inicial da disponibilização dos contêineres, de tal forma que a quantidade de *VMs* em uso seja a mínima possível, e de que os requisitos mínimos para execução da aplicação sejam supridos. O segundo objetivo é garantir que o componente auto escalonador do orquestrador também respeite as mesmas regras definidas para disponibilização inicial das cargas de trabalho, e assim, novas *VMs* só sejam provisionadas conforme necessário. E o terceiro objetivo envolve a otimização do componente responsável pelo reajuste dos contêineres (*re-schedule*), para realizar a consolidação das aplicações na menor quantidade de *VMs* possível. O conjunto destas três tarefas teve como intuito a redução dos custos operacionais com o *cluster* computacional, ajustando-o de forma dinâmica conforme a demanda das aplicações. A proposta de implementação deste escalonador obteve uma redução de 58% em comparação à implementação padrão.

(JAMES; SCHIEN, 2019) descrevem um projeto para implementação de um processo de orquestração com foco na utilização de *datacenters* com baixa emissão de carbono, realizando a implantação e migração das cargas de trabalho para localidades com menor intensidade de carbono na eletricidade. O ranking dos países com a maior emissão de carbono foi elaborado utilizando como base as localidades em que os grandes fornecedores de computação em nuvem operam (*Microsoft*, *Google*, *Amazon* e *Oracle*), e realizando o monitoramento da intensidade na emissão de carbono em um intervalo de tempo pré-definido. Os resultados obtidos foram empíricos, apresentando apenas uma classificação das cargas de trabalho dispostas entre diferentes *datacenters* durante horários específicos do dia, não sendo apresentado nenhum valor referente a redução de CO_2 resultante com a aplicação desta estratégia.

(ROCHA et al., 2019) elaboraram um algoritmo para disponibilização dinâmica das cargas de trabalho em um *cluster* computacional heterogêneo, denominado *HEATS* (*Heterogeneity and Energy-Aware Task-based Scheduling*), buscando a melhor relação entre desempenho e consumo de energia. A solução possui um componente responsável por realizar o monitoramento do desempenho e o consumo de energia dos sistemas hospedeiros. Conforme as aplicações vão sendo disponibilizadas, inicia-se o monitoramento e migração das mesmas para diferentes nós computacionais disponíveis, até que seja encontrada a compatibilidade mais eficiente, considerando a relação de desempenho e o consumo energético. O modelo de orquestração proposto nesta abordagem obteve redução no consumo de energia de 8,5%, enquanto que o impacto durante a escolha do nó computacional mais indicado para a instância computacional teve aumento de 7% no tempo total necessário para execução da tarefa.

(ROSSI et al., 2020) apresentam um algoritmo que implementa adaptação dinâmica e monitoramento de rede, baseado em latência, durante a orquestração de contêineres,

denominado *ge-kube* (*Geo-distributed and Elastic deployment of containers in Kubernetes*). A partir de parâmetros mínimos de desempenho esperados da aplicação relacionados a latência, novas instâncias da aplicação são disponibilizadas de forma dinâmica para atender a demanda. Como resultado, o trabalho propõe um benefício qualitativo frente a implementação padrão oferecida pelo *Kubernetes*, garantindo uma melhor distribuição das instâncias computacionais, respeitando requisitos de qualidade na prestação do serviço e beneficiando aplicações sensíveis a oscilações de rede.

O estado da arte na orquestração de contêineres não apresenta soluções para cenários em que se deseja aplicar um direcionamento específico às cargas de trabalho, além disso, nenhum leva em consideração estratégias referente a requisitos de negócio ou legais, abrindo espaço para serem melhor exploradas. Do ponto de vista técnico, os trabalhos não fornecem subsídios para exploração de sua implementação, com o objetivo de continuidade e melhoria do escalonador proposto ou simplesmente a sua reprodução em ambiente local, seja pelo fato de terem sido implementados em ambientes de simulação ou não oferecerem acesso público ao seu respectivo binário ou código-fonte. Por fim, também não ficou claro nos trabalhos a possibilidade de operação simultânea de um ou mais escalonadores, inclusive com o padrão oferecido pelas ferramentas de orquestração, dificultando a análise sob a possível operação em um ambiente orquestrado com diferentes necessidades relacionadas a regras de escalonamento.

3.2 Orquestração *Multi-Cloud*

Dentre as estratégias existentes para implantação e disponibilização de cargas computacionais em nuvem, está a chamada de *multi-cloud*, utilizada para realizar a distribuição das aplicações através de vários ecossistemas distintos. Sua popularização se deve ao fato do crescimento de estratégias voltadas à portabilidade e flexibilidade na execução das aplicações em nuvens públicas e privadas.

Outra característica a ser explorada em *multi-cloud* é a grande variedade na oferta de produtos e serviços, em que cada fornecedor tende a seguir e oferecer uma linha de soluções computacionais para se diferenciar de seus concorrentes, e assim, atrair a atenção e interesse na contratação de seus serviços. Neste ponto, a orquestração se torna fundamental, pois é através dela que será possível, durante a etapa de *design* da aplicação, realizar a definição dos locais de disponibilização da aplicação, garantindo a afinidade dos recursos dependentes para sua execução com seus respectivos fornecedores (BALALA, 2018).

3.2.1 Trabalhos Relacionados

A literatura aborda o papel fundamental que os orquestradores têm sob a gestão de um ecossistema *multi-cloud*, e também, as possibilidades de personalização durante o

processo de distribuição das cargas computacionais. As publicações abordam diferentes cenários onde a orquestração tem papel chave no processo *multi-cloud*, desde as facilidades oferecidas aos profissionais pelas ferramentas de orquestração, como a documentação, camadas de gestão, APIs e *dashboards*, até a escolha de fornecedores baseado no desempenho das aplicações, buscando a melhor relação entre desempenho e custo, balanceamento entre multi localizações geográficas para garantir o maior índice de disponibilidade possível, aproveitamento de políticas de preço mais competitivas.

(TOMARCHIO; CALCATERRA; MODICA, 2020) realizam uma revisão sistemática sobre as melhores práticas da orquestração de recursos heterogêneos através de múltiplos fornecedores, utilizando os chamados *CROF (cloud resource orchestration frameworks)*, realizando comparativos entre os mais relevantes. O conceito de interoperabilidade utilizado em computação em nuvem se refere a capacidade de movimentar as cargas de trabalho e dados entre os fornecedores, diante disso, organizaram como características mais relevantes os seguintes aspectos: ferramentas de acesso (CLI, API, Dashboards); gestão das aplicações e seu ciclo de vida; gestão de recursos, sua provisão e monitoramento; e; gestão da disponibilização das cargas de trabalho, como sua criação, inicialização, elasticidade, interrupção e remoção. Além disso, destacam a importância no suporte a múltiplos fornecedores, possibilitando a escolha pelo melhor dentre os disponíveis a partir de parametrização prévia da aplicação, além disso, garantir uma seleção sofisticada do fornecedor de acordo com os componentes necessários para execução da aplicação, otimização de custos, melhoria na qualidade do serviço e proteção a falhas operacionais de um fornecedor ou *datacenter*.

(CARVALHO; ARAUJO, 2020) apresentam um estudo a respeito de ferramentas utilizadas na gestão de ecossistemas *multi-cloud*, também conhecidas como *cloud orchestrators*. O objetivo foi identificar o desempenho dentre as ferramentas pesquisadas em relação ao processo de configuração do ecossistema, consumo de recursos e temporizadores de todas as etapas do processo. Como resultado, chegaram a dados que indicam a ferramenta *Terraform*[1] como a mais eficiente durante o processo, oferecendo um alto grau de maturidade, devido ao seu índice de compatibilidade com mais de 30 fornecedores de nuvem computacional.

(RAJ; RAMAN, 2018) fazem uma análise ampla sobre os pontos relevantes envolvendo o conceito de *multi-cloud*, abordando detalhes sobre a sua finalidade, processos envolvidos durante a orquestração de cargas de trabalho sob este ecossistema, oportunidades e possibilidades na sua implementação, modelos de disponibilização de aplicação, estratégias de alta disponibilidade (*cold standby, warm standby, hot standby* e *hot standby with read replicas*) e desafios relacionados a *data lock-in* e *vendor lock-in*, suas barreiras técnicas, operacionais e de negócio.

[1] <https://www.terraform.io>

(TORDSSON et al., 2012) exploram o conceito da heterogeneidade da computação em nuvem sob o paradigma de múltiplos fornecedores de nuvens computacionais e diferentes configurações nós (pequenos, médios, grandes e extra grandes). Como resultado, apresentam dados que confirmam a eficiência de um ecossistema *multi-cloud*, baseado em uma maior obtenção de desempenho sob menores custos operacionais, se comparado a utilização de uma nuvem única. Utilizando um conceito de *broker*, abordam tarefas relacionadas a definição do melhor local para disponibilização de uma carga computacional, dentre um conjunto de fornecedores, e, realiza a gestão e monitoramento destas cargas computacionais. Com o apoio de um componente *scheduler*, analisa periodicamente o ecossistema para planejar a distribuição das novas cargas de trabalho. Na conclusão do trabalho não foi especificado números absolutos referente a redução econômica na adoção desta estratégia, mas uma abordagem sobre os benefícios na utilização de nós computacionais com maior quantidade de recursos para realizar tarefas que exijam maior poder de processamento, e com isso, levam um tempo menor para concluir a execução, remetendo a um menor investimento frente ao valor/hora de processamento.

(GUERRERO; LERA; JUIZ, 2018) propõem uma otimização do processo de disponibilização de micro serviços em um ecossistema *multi-cloud*, a partir de: custos do fornecedor, latência de rede entre os micro serviços e tempo para início de uma nova instância quando uma outra se torna indisponível. A contribuição proposta está relacionada a melhorias na orquestração de contêineres em um ecossistema *multi-cloud*, validando-a através de comparativos ao desempenho do mesmo processo realizado com os algoritmos *Greedy First-Fit* e *Non-dominated Sorting Genetic Algorithm II (NSGA-II)*. O resultado obtido foi um desempenho melhor em 309% na utilização no *NSGA-II* se comparado ao algoritmo *Greedy*.

(MORENO-VOZMEDIANO et al., 2018) apresentam um método de orquestração para automatizar a disponibilização e gestão de aplicações, com características de alta disponibilidade em múltiplos fornecedores de computação em nuvem, implementando mecanismos de afinidade das cargas de trabalho, como a alocação de uma VM em um fornecedor específico. A implementação deste conceito de afinidade é garantida através de uma terminologia chamada de *role*, realizando a classificação de uma VM ou grupo de VMs para que sejam disponibilizadas em um local específico. A partir de dados referente a taxas de disponibilidade de fornecedores de nuvem computacional como AWS, IBM Softlayer, Digital Ocean, etc. apresentam uma previsão de disponibilidade geral da aplicação executando sob este método de 99,999%.

(BAUR et al., 2018) propõem uma solução para vincular um fornecedor de nuvem computacional a restrições pré-definidas, e também, especificar requisitos necessários para uma aplicação. A partir destes dados, um algoritmo denominado CSP (*constraint satisfaction problem*) irá fazer a escolha sobre qual fornecedor será selecionado para receber

a aplicação, tendo como principal benefício abordado no estudo, o apoio em evitar o *vendor lock-in*. Os principais desafios nesta tarefa estão relacionados em disponibilizar uma linguagem declarativa para que os usuários possam especificar os requisitos para execução da aplicação, e, um recurso ou algoritmo para realizar o confronto das ofertas de serviços de cada fornecedor e os requisitos especificados para a aplicação. Os parâmetros sugeridos para a linguagem declarativa são: localização geográfica, sistema operacional (família, versão, arquitetura), núcleos de CPU, memória RAM e espaço de armazenamento.

O estado da arte da orquestração de cargas de trabalho em um ecossistema *multi-cloud* tem tido um foco relacionado a critérios técnicos, como a oferta de produtos e serviços, ou ainda, evitar uma dependência direta a um único fornecedor. Nenhum estudo até então abordou características de negócio, para satisfazer restrições comerciais ou legislação local, ligadas diretamente à privacidade dos dados.

Por mais que as ferramentas disponíveis para implementar um ambiente *multi-cloud* tenham atingido um bom nível de maturidade de suas funcionalidades, ainda existe espaço para otimização dos processos de disponibilização, re-disponibilização e auto escalonamento, utilizando para isso a personalização de componentes responsáveis por tais tarefas.

A Tabela 2 relaciona os trabalhos relevantes, mais especificamente aqueles cujo objetivo é a substituição do comportamento padrão utilizado pela ferramenta de orquestração durante a alocação das cargas de trabalho. Cada linha representa um trabalho de pesquisa, onde a coluna “Referência” se refere ao trabalho relacionado, a coluna “Ferramenta” referência a ferramenta de orquestração utilizada, a coluna “Tecnologia” contém a tecnologia alvo da orquestração, e por último, a coluna “Implementação”, que resume a estratégia de implementação do objeto alvo da pesquisa.

3.3 Considerações Finais

O crescimento na adoção da computação em nuvem tem aumentado a busca por ferramentas que possuam flexibilidade na escolha pelo comportamento adotado pelo orquestrador, com adaptação a modelos baseados em consumo energético, qualidade do serviço e requisitos legais. Esta ferramenta deve estar apta a receber a parametrização destes diferentes modelos e regras restritivas, ajustando a orquestração dos contêineres em tempo de execução (CASALICCHIO, 2017).

A personalização do orquestrador pode ser realizada com a implementação de algoritmos que atendam às necessidades específicas de um ecossistema. Como cada ambiente é altamente dinâmico, diferentes objetivos são esperados como resultado da orquestração, como a minimização dos custos operacionais, otimizar o uso dos recursos disponíveis, redução da energia consumida, planejamento orçamentário, atendimento a diferentes níveis de *SLAs*, requisitos de negócio, etc.

Referência	**Ferramenta**	**Tecnologia**	**Implementação**
(TORDSSON et al., 2012)	OpenNebula	VM	Ambiente experimental
(MORENO-VOZMEDIANO et al., 2018)	OpenNebula	VM	Proposta de algoritmo
(GUERRERO; LERA; JUIZ, 2018)	OpenNebula HashiCorp Nomad	Contêiner	Proposta de algoritmo
(BAUR et al., 2018)	Cloudiator	VM	Proposta de algoritmo
(ROCHA et al., 2019)	Kubernetes	Contêiner	Escalonador personalizado
(JAMES; SCHIEN, 2019)	Kubernetes	Contêiner	Escalonador personalizado
(ROSSI et al., 2020)	Kubernetes	Contêiner	Escalonador personalizado
Proposta	Kubernetes	Contêiner	Escalonador personalizado

Tabela 2 – Lista de trabalhos com propostas de implementação personalizada do processo de orquestração

A adaptação a um conjunto de políticas de orquestração complexas, que sobrescrevem o comportamento padrão dos orquestradores requer capacidades fundamentais, como a autocorreção para melhorar a disponibilidade das aplicações, auto-otimização para melhor consumo dos recursos disponíveis e segurança dos sistemas (CASALICCHIO, 2019).

Realizar o escalonamento e distribuição das cargas de trabalho em um ecossistema *multi-cloud* tem se tornado uma tarefa cada vez mais importante, e carrega consigo problemas a serem resolvidos, como a parametrização prévia de requisitos de negócio. O objetivo desta pesquisa está relacionado a como esta definição será realizada durante a etapa de configuração da carga de trabalho no ambiente orquestrador *multi-cloud*, oferecendo um conjunto de variáveis que poderão compor seus arquivos de parametrização e configuração, como a região geográfica onde a distribuição deve ser realizada e a afinidade com outras cargas computacionais para garantir o nível mínimo de latência na troca de informações com outros módulos do ecossistema.

Diferente de estratégias utilizadas em trabalhos encontrados no estado-da-arte, o escalonador proposto não tem o objetivo de atender a uma necessidade específica durante o escalonamento, mas permitir, através de rótulos, a condução e direcionamento das instâncias da aplicação à nós computacionais que atendam seus requisitos, fortalecendo sua característica de heterogeneidade.

Além disso, cada aplicação a ser disponibilizada no ecossistema *multi-cloud* pode ter sua configuração específica, respeitando seus próprios requisitos, ou ainda, utilizando a implementação padrão do orquestrador. Isto garante que o ambiente orquestrado não tenha a necessidade de trabalhar com múltiplos escalonadores personalizados, cada um com um objetivo. Os detalhes relacionados a esta implementação serão descritos no Capítulo 4.

4 Escalonamento personalizado de cargas de trabalho

A tarefa de escalonamento das cargas de trabalho existentes nas ferramentas de orquestração é fundamental. Ao considerar ecossistemas computacionais composto por aplicações heterogêneas, cada grupo de aplicações pode possuir regras comportamentais e estratégias diferentes durante o processo de distribuição. Estas regras podem estar relacionadas a requisitos computacionais físicos (como as quantidades mínimas de processamento, memória, armazenamento e arquitetura), volume e intensidade de tarefas computacionais de acordo com a demanda dos usuários ou requisitos de negócio definidos durante a fase de concepção da aplicação.

Os requisitos de negócio a serem considerados nessa fase são todos aqueles que influenciam sobre o local em que será realizada a disponibilização da aplicação, mais especificamente em relação ao nó computacional. Eles podem estar relacionados a limitações fiscais, orçamento, segurança e governança de dados, níveis mínimos de desempenho e disponibilidade de acesso, requisitos computacionais de *hardware* e *software*, dentre outros.

As estratégias adotadas pelas ferramentas de orquestração existentes são direcionadas para cenários estáticos de configurações, utilizando para isso algoritmos de aplicação geral, como *bin-packing*, *best-fit* e *first-fit* (KUBERNETES, 2020b). Estas estratégias não são exclusivas para cenários *multi-cloud*, mas para todos os casos em que estejam disponíveis mais de um nó computacional em um *cluster*.

Algumas propostas de algoritmos alternativos, e de propósitos específicos, são encontrados na literatura, como apresentado na Seção 3.1. Esses algoritmos podem ser aplicados em ecossistemas *single-cloud* ou *multi-cloud*, desde que estejam disponíveis mais de um nó computacional no *cluster*, pois visam melhorias no direcionamento de uma instância da aplicação para um nó computacional específico, após submetê-lo a novas variáveis durante o processo de escalonamento. Como visto anteriormente, foco na otimização e melhor aproveitamento da capacidade computacional ociosa, para que a quantidade de nós ativos no *cluster* seja o menor possível, e assim proporcionar uma redução do investimento financeiro (RODRIGUEZ; BUYYA, 2018); priorização de fornecedores que possuam infraestrutura com baixa intensidade na emissão de carbono (JAMES; SCHIEN, 2019); otimização da disponibilização de instâncias da aplicação em localização geográfica o mais próxima possível, a fim de reduzir a latência na comunicação entre seus componentes (ROSSI et al., 2020); melhor aproveitamento da oferta de serviços heterogêneos de diferentes fornecedores (TOMARCHIO; CALCATERRA; MODICA, 2020); escalonamento das instâncias da aplicação respeitando conjuntos de regras definidas

utilizando linguagem declarativa, através do uso de sintaxe declarativa, e assim, reduzir a dependência de fornecedores (BAUR et al., 2018).

Com a evolução dos modelos de escalonamento das aplicações em nuvens computacionais, aliado a heterogeneidade destas aplicações em um mesmo ecossistema, a implementação de regras utilizando características empíricas de cada aplicação, como requisitos de negócio, ficam impossíveis de serem atendidas pelas implementações fornecidas por padrão. Por outro lado, as soluções presentes na literatura não compreendem completamente tais requisitos de negócio, além disso, não permitem classificar as cargas de trabalho, impossibilitando o direcionamento para os nós apropriados.

4.1 Escalonamento por afinidade de rótulos

Em geral, as soluções abordadas no estado-da-arte propõem a aplicação de algoritmos focados na escolha objetiva de qual nó computacional satisfaz um determinado critério, e uma vez definido, constrói-se o escalonador que irá substituir a implementação padrão oferecida pela ferramenta de orquestração.

Com o objetivo de proporcionar um controle dinâmico, considerando requisitos que envolvam múltiplos objetivos, multi-usuário e *multi-cloud*, desenvolveu-se uma estratégia flexível para que seja possível realizar o escalonamento respeitando um conjunto pré-definido de variáveis, que por sua vez representam requisitos de negócio de cada aplicação, de tal forma que possibilite vincular as cargas de trabalho aos nós computacionais.

Ao considerar cenários onde existam requisitos de negócio que se estendem até o local de hospedagem e distribuição da aplicação, novas variáveis podem compor a estratégia de orquestração, sem abrir mão dos benefícios proporcionados pelas ferramentas de gestão *multi-cloud*. A definição destas novas variáveis são viabilizadas através da adição de rótulos aos componentes do ecossistema orquestrado, mais especificamente, nas cargas de trabalho e nós computacionais. Estes rótulos ficam armazenados como metadados destes componentes, sendo utilizados como apoio na sua classificação e organização, mas em nosso contexto, serão considerados durante as etapas de escalonamento das instâncias da aplicação.

Utilizando uma sintaxe simples, declarativa e com estrutura pré-definida, novas condições são adicionadas conforme a necessidade, com suporte a uso de operadores lógicos, critérios de opcionalidade e priorização de rótulos específicos de acordo com sua importância. A combinação destas variáveis possibilita atender cargas de trabalho com diferentes objetivos e requisitos, além de permitir que múltiplas regras de negócio sejam consideradas, sob gerenciamento de um único escalonador personalizado.

Nesse contexto, explorar o recurso de rótulos da ferramenta de orquestração permite a construção de um escalonador personalizado capaz de interpretar um conjunto de parâmetros prévios, realizados diretamente nas cargas de trabalho e nos nós computacionais, e assim, distribuir as aplicações de maneira direcionada. Com a implementação desta extensão ao componente de escalonamento, foi possível interceptar a tarefa de distribuição das aplicações, escolhendo o nó computacional respeitando requisitos de negócio, tais como, níveis mínimos de latência da aplicação, restrição no uso de fornecedores que utilizem energias renováveis, localização geográfica para conformidade com legislações regionais, planejamento orçamentário, entre outros. Além disso, esta abordagem permite, inclusive, que outras propostas encontradas no estado-da-arte sejam reproduzidas neste modelo, através da configuração prévia das cargas de trabalho de acordo com os requisitos da aplicação ou do negócio.

A Seção 4.2 apresenta uma visão geral do cenário de orquestração. A Seção 4.3 aborda o fluxo do processo padrão de escalonamento do orquestrador *Kubernetes*. Já nas Seções 4.4 e 4.4.1 é apresentada a estratégia para construção do escalonador personalizado utilizando como estratégia a definição de rótulos, para marcação de características das cargas de trabalho e nós computacionais. Por fim, a Seção 4.4.2 demonstra a utilização do escalonador personalizado e como acontece a definição na carga de trabalho para que o utilize.

4.2 Visão Geral

A Figura 8 apresenta a interação dos contêineres de usuários com os nós computacionais, sendo intermediados pelo escalonador personalizado, que realiza a distribuição e escalonamento nos nós computacionais disponíveis, utilizando regras pré-definidas.

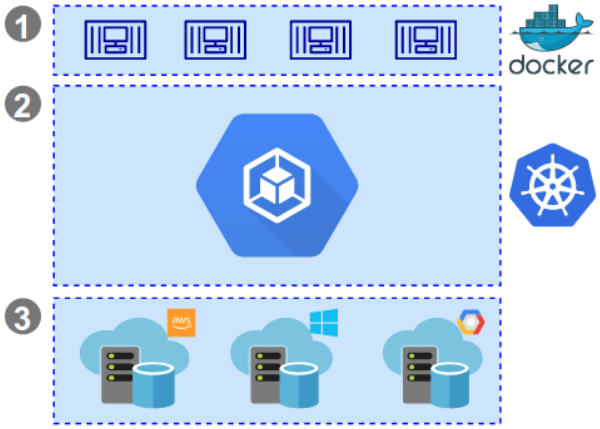

Figura 8 – Arquitetura de escalonamento personalizado de contêineres.

No bloco 1, no topo da figura, estão as aplicações conteinerizadas, que representam as cargas de trabalho que terão suas instâncias executadas no *cluster*. Conforme exposto na

Seção 2.3, em cada contêiner estão encapsuladas as aplicações junto com suas dependências, de tal forma que permita sua execução de forma independente. A ferramenta *Kubernetes* é especializada na orquestração das cargas de trabalho utilizando diferentes ecossistemas de contêineres, tais como: *Docker*, *CRI-O* e *Containerd*. Dentre todos estes, a escolha foi pelo *Docker*, por ser considerado o mais comum e abrangente, permitindo a construção de imagens das aplicações de forma rápida e fácil, reaproveitando componentes compartilhados pela comunidade.

O bloco número 2, representa o orquestrador de contêineres, responsável por realizar a tarefa de distribuição das instâncias computacionais nas máquinas virtuais, através de recursos para automação destas tarefas. Por meio da disponibilização de linguagem declarativa, permite a simplificação das operações de configuração e gestão de todo o ecossistema orquestrado, garantindo resiliência durante a execução das aplicações, através de mecanismos que garantem sua disponibilidade, e segurança em todo o processo, reduzindo a interação humana, mitigando assim a chance de erros. Como descrito no Capítulo 3, o orquestrador realiza as tarefas de seleção, implantação, monitoramento e controle dinâmico das cargas computacionais em um *cluster* computacional.

Por fim, o bloco número 3 representa os nós do *cluster* computacional, distribuídos entre os vários provedores de *cloud*. Cada instância da aplicação pode ser direcionada para um nó diferente, de acordo com o conjunto de regras definido. O objetivo desse trabalho flexibiliza a localização dos nós computacionais participantes do *cluster*, permitindo que os mesmos estejam distribuídos entre vários fornecedores distintos, caracterizando assim um ambiente *multi-cloud*. Esta tarefa de direcionamento possui vários objetivos e cenários para aplicação, conforme apresentado na Seção 2.2.3.

Para implementação destas funcionalidades, serão utilizados ecossistemas de computação em nuvem convencionais de mercado, envolvendo recursos de diferentes fornecedores, constituindo assim um *cluster multi-cloud* e espalhados globalmente de acordo com as ofertas de cada fornecedor, contendo máquinas virtuais que serão utilizadas para distribuir as aplicações, a partir do direcionamento das cargas de trabalho realizadas pelo orquestrador. Cada máquina virtual estará disponível no *cluster* computacional, sendo este administrado pelo orquestrador.

A seguir, são apresentados os detalhes técnicos do modelo de orquestração personalizada. Na Seção 4.3 é apresentada a ferramenta de orquestração de contêineres em ecossistema *multi-cloud Kubernetes*, evidenciando o seu fluxo de etapas para realizar a disponibilização das instâncias computacionais. Na Seção 4.4 é abordada a estratégia de implementação do orquestrador personalizado. Por fim, na Seção 4.4.3 é discutido sobre os cenários ideais para aplicação deste modelo de orquestração, suas restrições e limitações.

4.3 Estratégia Padrão de Escalonamento no *Kubernetes*

Os principais componentes em um ecossistema de orquestração *multi-cloud* são os nós participantes do *cluster*, os contêineres computacionais e suas instâncias da aplicação, chamadas de *pods*. Apresentados inicialmente na Seção 2.3.1, estes componentes são fundamentais durante o processo de escalonamento.

Cada nó computacional estará conectado ao gerenciador *master* do orquestrador, e é subordinado ao mesmo, passando a receber as instâncias das aplicações para execução conforme as regras de balanceamento definidas.

Os contêineres representam as cargas de trabalho, armazenando detalhes e parâmetros relacionados a sua execução, enquanto um *pod* representa a menor unidade de trabalho em um ambiente orquestrado, contendo uma cópia da aplicação e possuindo um isolamento a nível de *kernel* durante a sua execução. Para cada *pod* em execução, o *Kubernetes* irá executar outro contêiner, responsável por coletar informações de execução e monitoramento do serviços, além disso, a ferramenta de orquestração realiza a disponibilização e destruição constantemente destes *pods*, conforme eventos de escalonamento (expansão e contração) vão sendo disparados (MOUAT, 2015).

Cada *pod* possui de forma encapsulada todos os recursos necessários para execução da aplicação, como armazenamento e comunicação de rede, além disso, requisitos mínimos de *hardware* para a sua execução podem ser informados durante a configuração da carga de trabalho, servindo como base para que o orquestrador realize as tarefas de decisão sobre em qual nó computacional o mesmo será disponibilizado, e ainda, realizar o escalonamento do *cluster* computacional com a agregação de novos nós. Os *pods* são projetados para executar uma única instância da aplicação, permitindo o escalonamento de múltiplas instâncias de forma horizontal[1].

Como principal tarefa, o *scheduler* supervisiona a criação de *pods*, em busca de qual nó computacional o mesmo será distribuído. Cada um destes *pods* podem possuir diferentes parâmetros para execução, e assim, os nós computacionais candidatos a recebê-los devem ser filtrados em busca da melhor compatibilidade.

O comportamento padrão do *Kubernetes* durante as tarefas de escalonamento dos *pods* leva em consideração o total de recursos computacionais disponíveis nos nós, optando sempre por aquele que esteja mais ocioso. Dentre as tarefas realizadas pelo escalonador, está o processo de redimensionamento das cargas de trabalho, executado a partir do monitoramento periódico de todos os *pods* disponibilizados. Este processo tem o objetivo de readequar a quantidade dos *pods* de acordo com o consumo de recursos computacionais por cada um deles, se for necessária a alocação de um novo *pod* de uma determinada carga

[1] disponibilização de novas instâncias da aplicação para balancear as requisições de usuários, sem realizar a adição de novos nós ao *cluster* computacional

de trabalho, todos os nós do *cluster* são analisados em busca daqueles que estejam com capacidade ociosa, porém, se for necessária uma redução da quantidade destes *pods*, o orquestrador realiza a destruição do mesmo.

Esta tarefa realizada pelo *Kubernetes* é garantida através do componente *HorizontalPodAutoscaler*, implementado como um laço contínuo de execução, com interações realizadas a cada 15 segundos, e que pode ser personalizado através da definição de parâmetro (KUBERNETES, 2019a). A Fórmula 4.1 apresenta o cálculo realizado pelo orquestrador durante a definição da quantidade de *pods* a serem disponibilizados para uma carga de trabalho, seu resultado será direcionado ao escalonador, o qual realizará o processo de alocação aos nós computacionais disponíveis.

$$instanciasDesejaveis = \lceil instanciasAtuais \times \left(\frac{consumoAtual}{consumoDesejavel}\right) \rceil \tag{4.1}$$

onde, "*instanciasAtuais*" é a quantidade atual de *pods* em execução de uma determinada carga de trabalho; "*consumoAtual*" é o consumo atual de recursos computacionais que estão sendo exigidos pelos *pods* em execução; "*consumoDesejavel*" é o consumo desejável de recursos computacionais, considerando a quantidade de *pods* em execução e a parametrização de recursos realizada em sua configuração; e; "*instanciasDesejaveis*" é a quantidade de *pods* desejáveis calculada para a carga de trabalho.

O resultado final obtido da Fórmula 4.1, atribuído à variável "*instanciasDesejaveis*", será enviado ao escalonador para alocação das novas instâncias.

A Figura 9 apresenta os componentes internos implementados pela arquitetura da ferramenta *Kubernetes* para realizar o monitoramento e gestão das responsabilidades durante o processo de escalonamento das cargas de trabalho. Para interagir com tais componentes e realizar o monitoramento dos eventos em cada etapa da disponibilização das aplicações, a ferramenta *Kubernetes* disponibiliza uma interface de integração com sua arquitetura baseada em uma *API REST*.

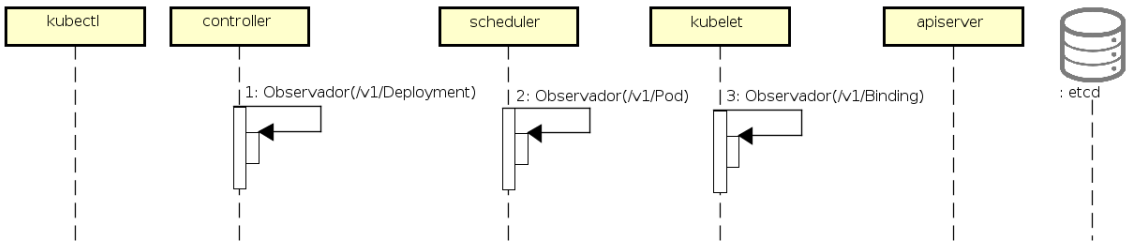

Figura 9 – Diagrama de Sequência contendo os componentes de controle da ferramenta *Kubernetes*. Adaptado de (KUBERNETES, 2020a)

O componente *controller* realiza o monitoramento da situação atual de cada tarefa de implantação, realizando ações necessárias para garantir uma execução bem sucedida.

O componente *scheduler* executa a regra de negócio responsável por realizar a busca e escolha do nós computacional mais aderente para implantação dos *pods*. O componente *kubelet*, em execução em cada nó computacional, recebe os eventos de vínculo, execução e disponibilização de um *pod*. O *apiserver* possibilita que diferentes processos, tanto internos quanto externos ao orquestrador *Kubernetes*, realizem interações durante todo o ciclo de vida da execução das cargas computacionais, permitindo que os eventos sejam interceptados e extensões personalizadas sejam construídas. Por fim, o componente *kubectl* é a linha de comando padrão utilizada para realizar a gestão do *cluster* computacional.

Na Figura 10 é apresentado o processo realizado durante a criação de um *pod*, onde o componente *apiserver* notifica a necessidade de um novo *pod*, seja por um processo de distribuição de uma nova carga computacional, escalonamento de nova instância para uma carga computacional já existente ou redistribuição em caso de falha. O componente *controller* por sua vez, através de seu evento de observador da fila de implantação inicia o processo de criação do novo *pod*.

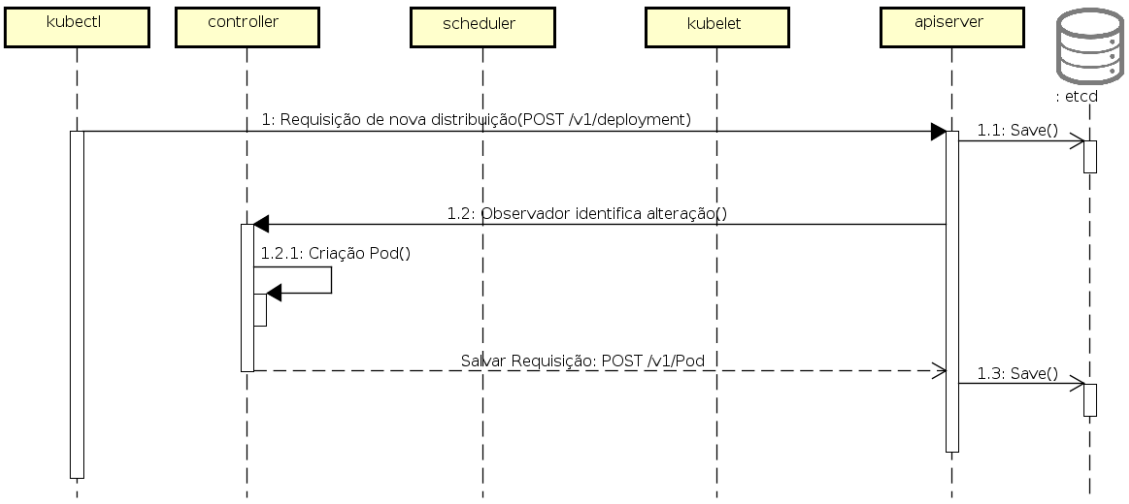

Figura 10 – Diagrama de Sequência referente ao processo de criação de um novo *pod*. Adaptado de (KUBERNETES, 2020a)

A etapa final, responsável por atribuir o *pod* recém criado a um nó computacional é representada na Figura 11, nela, o componente *apiserver* notifica o componente *scheduler* sobre a necessidade de vínculo do *pod* recém criado a um nó computacional, este por sua vez, aplica a regra padrão de escalonamento, ou, aplica regra personalizada se houver.

O processo de escalonamento padrão da ferramenta *Kubernetes* possui duas etapas básicas, que acontecem no nó computacional principal do *cluster*, denominado *master*. Na primeira delas, todos os nós computacionais disponíveis são selecionados, e aqueles que não estejam em um estado válido ou que não tenham recursos livres suficientes para receber a nova instância são desconsiderados. A segunda etapa consiste em um processo

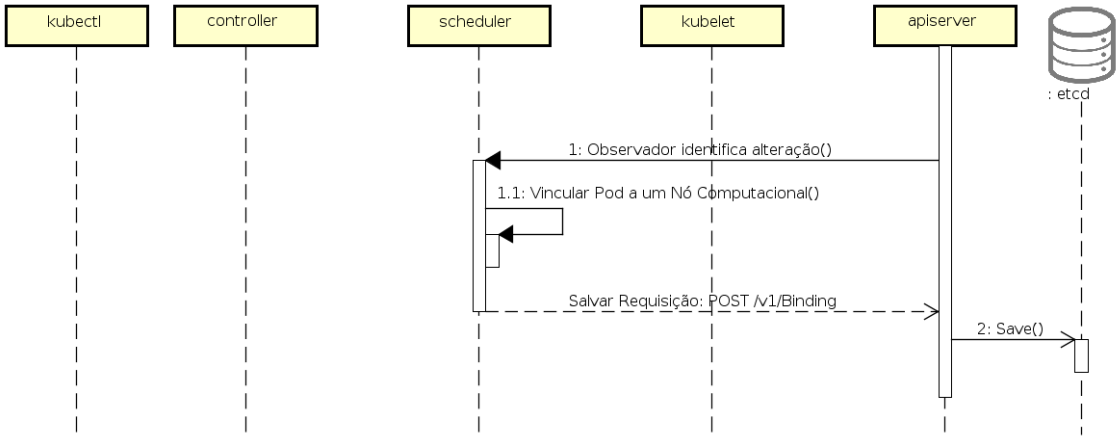

Figura 11 – Diagrama de Sequência referente ao processo de atribuição de um *pod* a um nó computacional. Adaptado de (KUBERNETES, 2020a)

de pontuação e classificação a partir de uma lista de prioridades, passando por três etapas básicas: *Predicate*, *Priority* e *Select*.

A Figura 12, detalha as atividades do escalonador padrão do *Kubernetes*, realizada durante a segunda etapa citada anteriormente, em que será feito o vínculo do *pod* ao nó computacional, também conhecido como *NodeBinding*. No bloco número 1 está representada a execução do componente padrão de escalonamento, onde é realizado o monitoramento constante dos *pods* que foram criados e que ainda não possuem nenhum nó computacional definido. Neste momento, o escalonador passa a ser responsável por estes *pods*, dando início ao processo compreendido pelas três etapas para a escolha do nó alvo da disponibilização de cada um destes *pods*. Já o bloco número 2 representa a etapa inicial da escolha do nó, chamada de *Predicate*. Nela são verificados todos os nós disponíveis e que não estejam sofrendo nenhuma "pressão" por falta de recurso físico para receber um novo *pod*. Como o escalonador padrão não verifica a ociosidade de cada nó a todo momento, o conceito de análise de "pressão" existe para identificar se algum destes nós não está passando por alguma falta de recurso (ex.: CPU, memória e espaço de armazenamento), e assim, seja desconsiderado desta etapa.

O próximo passo é a etapa de *Priority*, representada no bloco número 3, onde a prioridade de escolha do nó computacional é realizada sob o aquele que possui a maior quantidade livre de recurso computacional, de acordo com a Fórmula 4.2, garantindo assim uma distribuição balanceada.

$$
\begin{aligned}
&tipo \in \{gpu, cpu, mem\} \\
&pontuacao = \sum \left(\frac{tipo((capacidade - \sum requisicoes) \times 10)}{capacidade} \right) /3
\end{aligned}
\tag{4.2}
$$

Figura 12 – Atividades realizadas pelo escalonador padrão do *Kubernetes*. Adaptada de (CASQUERO et al., 2019)

onde, "*tipo*" é a pontuação do nó computacional, equivalente aos seus recursos físicos, como: memória, CPU e GPU; "*capacidade*" a capacidade computacional do nó; "*requisicoes*" é a soma das requisições de todos os *pods* ao nó computacional; e; "*pontuacao*" é o resultado obtido da fórmula, se referindo ao nó computacional com a maior fração livre de recursos. O resultado da Fórmula 4.2 será uma fração entre 0 e 10, onde 10 é o nó computacional mais indicado para recepcionar o *pod*, e, se houverem dois ou mais nós com a mesma pontuação, a escolha acontece de forma aleatória.

Por fim, a etapa *Select*, localizada no bloco número 4, que é responsável por retornar o nó que possui a maior pontuação, e que será o escolhido para executar a instância da aplicação (WANG et al., 2021). Como há uma interação constante do *Kubernetes* e o componente de *scheduler*, cada tarefa que afete um *pod* é submetida ao componente *custom scheduler*, informando a qual se refere, qual situação o mesmo se encontra (pendente de disponibilização, recursos computacionais insuficientes ou interrupção na execução) e a lista de nós disponíveis no *cluster* computacional. Esta ação possibilita a escolha pelo nó computacional ao qual o *pod* será direcionado, permitindo que um conjunto diferente de políticas sejam consideradas, e ao término, retornar ao orquestrador qual o nó foi definido.

Se por algum motivo durante o processo de *scheduling* nenhum nó for definido para receber a distribuição do *pod*, o mesmo não será disponibilizado e ficará em situação

pendente. Isto pode ocorrer pela ausência de recursos computacionais físicos disponíveis, ou em nosso caso, nenhum nó atender aos parâmetros personalizados da carga computacional, como restrição a uma determinada localização geográfica ou afinidade com outro módulo da aplicação.

Considerando que o escalonamento do *Kubernetes* não agrega os requisitos de negócios já discutidos anteriormente, esse trabalho propõe a personalização do componente *custom scheduler*. Realizando a sobreposição de sua implementação padrão, estendendo as funcionalidades do *kube-scheduler* (componente padrão de escalonamento), através do monitoramento do processo de escalonamento utilizando sua *API REST*. Isto permitirá que sejam agregadas regras de negócio que estão relacionadas a requisitos do projeto alvo da orquestração das cargas computacionais.

Na Seção 4.4 serão detalhadas quais as configurações que estarão disponíveis para utilização, e como será realizada a parametrização de cada nó computacional e as cargas de trabalho no ecossistema orquestrado.

4.4 Escalonador Personalizado para o *Kubernetes*

A personalização do componente padrão de escalonamento tem como objetivo permitir, através de linguagem declarativa, atribuir características às cargas de trabalho e aos nós computacionais, de tal forma que a etapa de escalonamento possa utilizá-las, realizando a combinação e direcionamento das instâncias.

O escalonador implementado tem por finalidade permitir que novas variáveis, baseadas nas regras de negócio, sejam adicionadas conforme a necessidade, sempre se atentando ao ponto de que seu objetivo fim está relacionado em vincular as cargas de trabalho aos nós computacionais de maior afinidade, mediante comparação e validação de marcações declarativas nestes dois componentes.

O processo de parametrização destas políticas, nestes componentes - nós computacionais e cargas de trabalho - utilizará o conceito de *label* da ferramenta *Kubernetes*, o qual permite que um conjunto de chave/valor seja atribuído a diferentes objetos de sua arquitetura. Sua utilização é indicada quando se pretende realizar alguma organização ou classificação de objetos em que não haja uma semântica padrão para isso. Estas informações podem ser adicionadas aos objetos durante sua fase de criação ou adicionadas e modificadas após já estarem em execução.

Utilizar *labels* nos componentes do *Kubernetes* permite criar estruturas aos objetos do orquestrador de forma organizada, não sendo necessárias ferramentas à parte para estes mapeamentos. A implementação deste recurso permite que durante a disponibilização de serviços seja possível um gerenciamento hierárquico e compartilhamento de informações

entre diferentes componentes do orquestrador, especialmente em infraestruturas rígidas, sem a interação dos usuários (KUBERNETES, 2020c).

Para exemplificar o uso das marcações personalizadas durante a configuração do ecossistema computacional, são sugeridas duas variáveis para estudo de caso[2], a primeira delas tem como objetivo o uso de energia verde durante a execução das instâncias da aplicação, e está relacionada ao crescente interesse das empresas e compromissos ligados à preservação ambiental. Já a segunda possui como objetivo a possibilidade da definição do local de distribuição com foco na redução da latência durante o acesso das aplicações, ou ainda, política de segurança e privacidade dos dados, que estão suscetíveis a legislação do local onde ocorre a prestação do serviço e que os usuários estão baseados, ou ainda, da legislação do local onde estão hospedados os dados.

Isto requer a classificação de cada nó participante do *cluster* com informações que o classifique quanto a:

1. Representatividade no uso de energia verde;
2. Localização geográfica.

A classificação de representatividade no uso de energia verde será realizada através da palavra reservada *green-energy-perc* e em sua parametrização poderá receber valores entre 0 e 100, de acordo com a representatividade deste tipo de fonte de energia no *datacenter* do fornecedor em que o nó computacional está sendo executado, onde 0 representa a não utilização de energia verde e 100 o consumo exclusivo deste tipo de fonte de energia. Grandes fornecedores de nuvem computacional pública como *Amazon Web Services*, *Google Cloud Platform*, *Microsoft Azure* e *Oracle Cloud* divulgam amplamente em seus canais de comunicação suas taxas e percentuais de representatividade, porém, pode ser necessária uma consulta direta ao fornecedor.

O classificador utilizado para definir a localização geográfica terá duas marcações distintas, uma com foco na região global em que o *datacenter* se encontra, e outra referência a sua localização regional, também conhecida como "zona", utilizando as palavras reservadas *dc-region* e *dc-zone* respectivamente. Estas informações utilizadas na parametrização devem levar em consideração dados obtidos junto ao fornecedor de nuvem computacional onde o nó está situado e não devem levar em consideração a média de todos os *datacenter* do fornecedor, mas sim a parametrização individual de cada nó computacional que façam parte do *cluster*.

Assim como feito para os nós, as cargas de trabalho também possuem uma marcação referente a representatividade mínima requerida no uso de energia verde, e também

[2] cenários escolhidos para exemplificar seu uso, não limitando a aplicação do orquestrador a apenas essas duas. Outros exemplos de cenários são apresentados no capítulo 5

relacionada à localização geográfica. Desta forma, durante a etapa de *NodeBinding*, em que um nó computacional é atribuído a uma instância que está sendo disponibilizada, é possível verificar qual é o mais indicado.

A definição destas políticas é realizada na configuração dos nós computacionais que fazem parte do *cluster*, através de *labels* que o representem, como a localização geográfica e seu nível de utilização de energia verde, e também das cargas de trabalho, que representam as aplicações que serão agendadas para execução pelo orquestrador, como a restrição de localização geográfica e a necessidade ou não de escolha por algum fornecedor que utilize energia verde, também através de *labels*. A cada realização de uma nova distribuição de instância da aplicação, nos processos de *schedule*, *re-schedule* e *auto-scale*, estas configurações são confrontadas, de tal forma que a aplicação seja distribuída em um nó computacional que satisfaça as políticas restritivas que foram definidas.

Para vincular diferentes *labels* para cada nó computacional, é necessário utilizar o comando *kubectl*, que permite a administração dos diferentes componentes da ferramenta de orquestração *Kubernetes*. O Comando 1 demonstra a adição de *labels* para categorizar um nó computacional.

```
kubectl label nodes <no-computacional> ppgcomp.unioeste.br/green-energy-perc=100
kubectl label nodes <no-computacional> ppgcomp.unioeste.br/dc-region=southamerica
kubectl label nodes <no-computacional> ppgcomp.unioeste.br/dc-zone=east1-a
```

Listagem 1: Adição de rótulos a um nó computacional.

As cargas computacionais também podem ser rotuladas, mas diferente do modelo de valores utilizados nos nós, podem ter informações específicas de cada parâmetro, podendo-se utilizar todas as funcionalidades da rotulagem estendida. A Listagem 2 mostra a adição de rótulos para categorizar uma carga de trabalho.

```
kubectl patch deployment <carga-trabalho> --type='json' -p='[{"op": "add", "path":
↪  "/spec/template/metadata/labels/ppgcomp.unioeste.br~1green_energy",
↪  "value":"ge-80"}]'
kubectl patch deployment <carga-trabalho> --type='json' -p='[{"op": "add", "path":
↪  "/spec/template/metadata/labels/ppgcomp.unioeste.br~1dc_region",
↪  "value":"eq-southamerica"}]'
kubectl patch deployment <carga-trabalho> --type='json' -p='[{"op": "add", "path":
↪  "/spec/template/metadata/labels/ppgcomp.unioeste.br~1dc_zone",
↪  "value":"like-_east_"}]'
```

Listagem 2: Adicionando rótulos a uma carga computacional.

Antes de disponibilizar cada instância da aplicação, o componente *custom scheduler* interceptará este evento, permitindo que seja realizado um confrontamento entre a parametrização de *labels* entre a carga de trabalho que está requisitando a disponibilização

de uma nova instância e os nós computacionais participantes do *cluster*, informados na requisição, e se houver disponibilidade, a instância em questão será direcionada.

O Algoritmo 1 detalha a regra de pontuação que definirá o nível de afinidade entre o *pod* e o nó computacional, a partir da lista de *labels* de cada um deles. Neste cenário a pontuação possível será entre 0 e 100, onde 0 representa a inexistência de qualquer *label* compatível entre eles, e 100 representa a compatibilidade total.

Algoritmo 1: Pontuação de afinidade entre os rótulos.

```
Input: podLabels: Lista dos rótulos vinculados ao POD
Input: nodeLabels: Lista dos rótulos vinculados ao nó computacional
Output: Pontuação percentual de afinidade obtida entre os rótulos
1  Function matchValue(podLabels, nodeLabels):
2      qtyNodeLabelMatch ← 0
3      podLabelSize ← podLabels.size()
4      foreach podLabel in podLabels do
5          nodeLabelMatch = nodeLabels.any (nodeLabel → nodeLabel.key ==
             podLabel.key)
6          if nodeLabelMatch.value == podLabel.value then
7              qtyNodeLabelMatch++
8          end
9      end
10     return (qtyNodeLabelMatch / podLabelSize * 100)
```

O Algoritmo 2 detalha como é o processo de atribuição de um nó computacional a um *pod*, percorrendo todos os nós computacionais disponíveis, extraindo sua informação referente a definição personalizada de *labels* e submetendo ao Algoritmo 1, que fará o cálculo do percentual de "afinidade" entre eles. Ao final, o nó computacional com a maior pontuação obtida será aquele considerado para receber a instância computacional da carga de trabalho.

Além disso, são abordados meios para que seja possível definir uma variável como desejada, porém não impeditiva, isto é, priorizar os nós computacionais com a maior afinidade com a instância computacional alvo da implantação, mas, se não houver nenhum disponível, direcionar para aquele mais próximo do desejado. O objetivo desta funcionalidade será garantir que um *pod* não fique sem nenhuma definição de nó computacional, caso contrário ele entrará em modo de "falha", indicando a falha durante o processo de escalonamento.

Algoritmo 2: Escolha do nó computacional de maior afinidade.

```
   Input: pod: POD que está em processo de implantação
   Input: nodes: Lista de nós computacionais disponíveis para implantação
   Output: node: Nó computacional para distribuição do POD
 1 Function podNodeBind(pod, nodes):
 2     if nodes == null then
 3         return null
 4     end
 5     nodes ← metricsserver.NodeMetricses()
 6     podLabels ← pod.labels
 7     nodeForPodBind ← null
 8     greatestNodePriority ← 0.0
 9     foreach node in nodes do
10         nodeLabels ← node.labels
11         podMatchValue ← matchValue(podLabels, nodeLabels)
12         if nodeForPodBind.matchValue <= podMatchValue and
            nodeForPodBind.nodePriority > greatestNodePriority then
13             nodeForPodBind ← node
14             greatestNodePriority ← nodeForPodBind.nodePriority
15         end
16     end
17     return nodeForPodBind
```

A Figura 13 ilustra o processo de atribuição de uma instância de aplicação a um nó computacional considerando os rótulos definidos diretamente na carga de trabalho. Estes rótulos são confrontadas com aquelas existentes no nó computacional, de tal forma que a escolha sobre qual deles receberá a aplicação seja o resultado daquele que possua a maior "afinidade" às restrições pré-definidas.

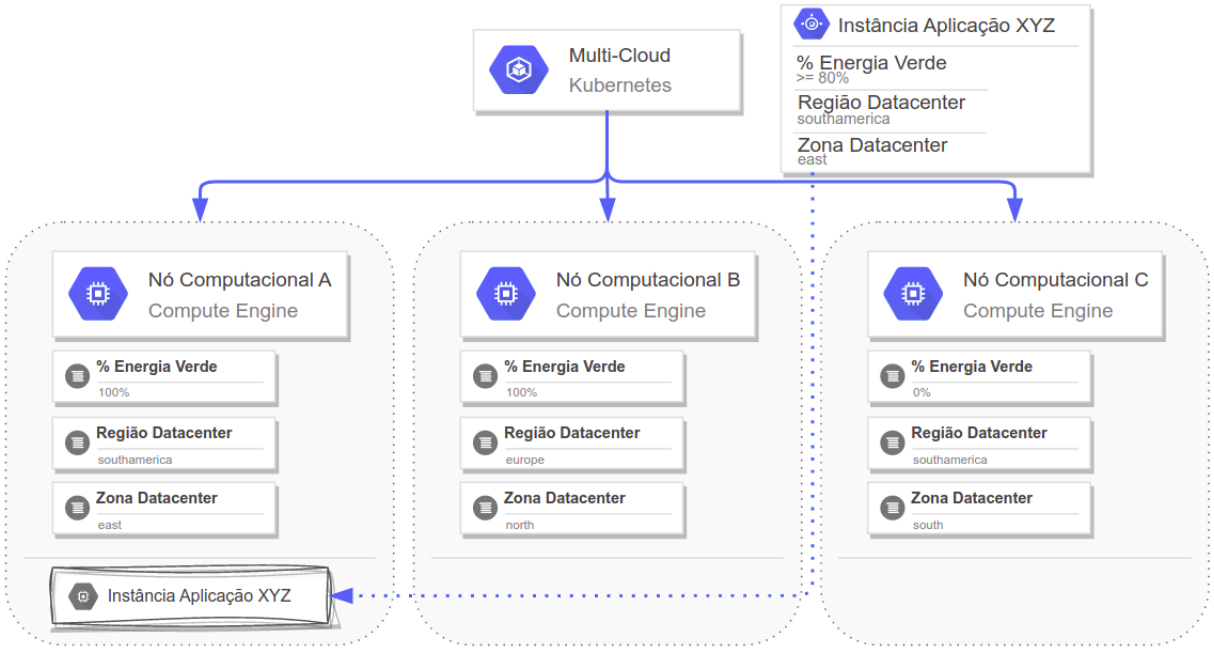

Figura 13 – Relacionamento entre cargas de trabalho e nós computacionais através de *labels*.

4.4.1 Rótulos personalizados

No *kube-scheduler*, os rótulos chave/valor de um *pod* e de um nó devem ser idênticos para viabilizar uma alocação. No escalonador personalizado implementado, estendemos a capacidade de rotulagem para permitir configurações mais flexíveis e amplas.

A primeira característica adicionada é a possibilidade da aplicação de operadores relacionais, de correspondência e de contenção, juntamente com a definição de valores para cada um deles, permitindo um controle mais granular quanto ao comportamento de aplicabilidade de cada um destes valores. A Tabela 3 detalha os operadores implementados e seus respectivos comportamentos.

Tabela 3 – Operadores adicionados ao esquema de rotulagem padrão do Kubernetes.

Grupo	**Operador**	**Comportamento**
Relacional	eq	Igual
Relacional	ne	Não igual
Relacional	gt	Maior que
Relacional	ge	Maior ou igual que
Relacional	lt	Menor que
Relacional	le	Menor ou igual que
Correspondência	like	Correspondência a valores
Correspondência	notlike	Não correspondência a valores
Correspondência	contains	Contêm valores
Correspondência	notcontains	Não contêm valores
Contenção	in	Contido em uma coleção de valores
Contenção	notin	Não contido em uma coleção de valores

Junto aos operadores relacionais, também define-se a opcionalidade do atributo, que é expressado com o uso do caractere "_" logo após a definição do operador. Na Listagem 3 apresenta-se um exemplo de aplicação dos rótulos obrigatórios e opcionais. Na linha 1 tem-se um atributo obrigatório e operação de igualdade ("eq") para o rótulo; na linha 2 é definido que o valor para o atributo seja maior que ("gt") 80, porém, opcional.

```
1 ppgcomp.unioeste.br/dc-zone=eq-east1-a //obrigatório
2 ppgcomp.unioeste.br/green_energ=gt_-80 //opcional
```

Listagem 3: Definição de rótulos obrigatórios e opcionais.

É possível observar também o uso de prefixos, permitindo que vários possam ser utilizados simultaneamente e de modo isolado, sem que haja conflito entre os rótulos gerenciados por cada um deles. Foi considerado o uso de sub-domínios *DNS* para prefixar aqueles que fazem parte de um mesmo contexto, (KUBERNETES, 2020c). Nos exemplos, utilizou-se o prefixo "*ppgcomp.unioeste.br*" para os rótulos.

Por fim, uma terceira característica foi incluída para permitir a classificação dos rótulos por nível de prioridade. Ela será usada nos casos em que houver empate por afinidade na compatibilidade dos rótulos, será analisado quais possuem a maior prioridade para *pod*, e assim, direcionar para o nó computacional mais adequado. A definição deste valor é realizada ao final do valor do rótulo, separado pelo caractere "." (ponto), que deve ser seguido por um valor numérico, positivo e válido. A Listagem 4 ilustra a definição de três rótulos com níveis de prioridade distintos. Neste cenário, um maior peso é atribuído ao rótulo da linha 2. A rotulagem definida para os nós computacionais e cargas de trabalho são armazenadas em suas respectivas entidades, dentro da ferramenta de orquestração, acessíveis para manipulação através de uma estrutura de arquivos no formato *YAML* (KUBERNETES, 2020d).

```
1 ppgcomp.unioeste.br/dc-vendor=in-azure-aws-gcp.10
2 ppgcomp.unioeste.br/dc-region=eq-southamerica.50
3 ppgcomp.unioeste.br/vm-so=linux.20
```

Listagem 4: Definição de diferentes níveis de prioridade aos rótulos.

4.4.2 Implementação e uso do escalonador proposto

A implementação do escalonador personalizado foi realizada utilizando a linguagem Go (nativa do Kubernetes) em sua versão 1.16. Também foram utilizadas um conjunto de bibliotecas, apresentadas na Tabela 4, para fornecer suporte e acesso aos componentes de gerenciamento de seus módulos internos, bem como manipulação dos rótulos personalizados.

Tabela 4 – Bibliotecas utilizadas na implementação do escalonador personalizado.

Biblioteca	**Versão**	**Objetivo**
k8s.io/client-go	0.21.1	Comunicação com componentes internos do Kubernetes
k8s.io/apimachinery	0.21.1	Acesso à informações de objetos internos do Kubernetes
k8s.io/metrics	0.21.1	Extração de métricas referente ao consumo computacional dos nós
github.com/minio/wildcard	1.0.4	Análise de correspondência de textos utilizando caracteres coringas

Para construção do escalonador utilizou-se alguns componentes internos do orquestrador Kubernetes com a finalidade de extrair informações de cada nó computacional. O componente *Node Informer* permite acessar detalhes quanto às marcações de rótulos definidas aos nós, quantidade de instâncias de aplicações (*pods*) já vinculadas e a quantidade de recursos computacionais disponíveis. Pelo componente *Pod Informer* é possível acessar detalhes da instância computacional (*pod*) recebida para alocação à um nó, onde detalhes referentes às marcações de rótulos da carga de trabalho serão extraídas para execução do algoritmo de afinidade.

A utilização do escalonador personalizado é definida no arquivo de configuração da aplicação. A definição é feita por meio do atributo opcional chamado de *schedulerName*. Para que o mesmo tenha efeito, antes é necessário realizar a implantação do escalonador personalizado diretamente no ecossistema Kubernetes, conforme apresentado na Listagem 5. Na sequência, é necessário definir o uso do escalonador personalizado em cada carga de trabalho que se deseja aplicar o comportamento de afinidade por rótulos, conforme apresentado na Listagem 6.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: label-affinity-scheduler
spec:
  template:
    spec:
      containers:
        - name: label-affinity-scheduler
          image: lfaltran/label-affinity-scheduler:1.9
```

Listagem 5: Implantação do escalonador personalizado no Kubernetes.

Os códigos-fonte do escalonador, modelos de configuração das cargas de trabalho, bibliotecas utilizadas e demais materiais de apoio, estão disponíveis para download no repositório <https://github.com/lfaltran/label-affinity-scheduler>.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stress-cpu
spec:
  template:
    spec:
      schedulerName: label-affinity-scheduler
      containers:
      - name: stress-cpu
        image: vish/stress
        args: ["-cpus", "2"]
```

Listagem 6: Vínculo entre a carga de trabalho e o escalonador personalizado.

4.4.3 Considerações e Limitações

Esse capítulo teve como objetivo apresentar o uso de um escalonador personalizado para a ferramenta *Kubernetes*, utilizando para isso um recurso disponibilizado em sua própria arquitetura, os *labels*, para a classificação, marcação e organização das cargas de trabalho e nós computacionais.

A principal contribuição nesse sentido é permitir que as cargas sejam alocadas conforme afinidade, realizando o confrontamento das marcações definidas durante a construção do ecossistema *multi-cloud*, classificação pela maior pontuação obtida e direcionamento ao nó computacional, realizando tal comunicação através de sua própria *API* embarcada.

O escalonador proposto também não leva em consideração a disponibilidade ociosa de recursos computacionais daqueles que infringirem alguma regra pré-definida, isto requer uma atenção durante o processo de configuração do ambiente, de tal forma que a disponibilidade de nós esteja apropriada para as restrições implementadas nas cargas de trabalho, caso contrário, as instâncias da aplicação não encontrarão um nó computacional compatível, e consequentemente, o escalonador não indicará ao orquestrador um nó computacional alvo para disponibilização.

A taxa de sucesso na alocação dos *pods* aos nós computacionais está ligada diretamente a três fatores fundamentais. O primeiro deles diz respeito a qualidade da definição dos rótulos a cada um dos nós computacionais que compõem o *cluster* orquestrado, isto é, garantir que todos os rótulos estejam corretamente preenchidos refletirá diretamente no direcionamento dos *pods* a cada um deles e o não infringimento de nenhum requisito funcional das aplicações.

Já o segundo fator tem relação a contratação de recursos computacionais que satisfaçam os requisitos funcionais das aplicações dos usuários, evitando assim que nenhuma aplicação fique sem um nó computacional em afinidade com ela. Por fim, outro cuidado que

precisa ser observado está na alta concentração de *pods* sob poucos nós computacionais, indicando uma necessidade de contratação de recursos computacionais extras com as mesmas características.

5 Experimentações e Resultados

Neste capítulo são apresentados diferentes cenários para a validação do escalonador personalizado. Tais cenários são usados tanto para analisar os resultados obtidos ao compará-lo à implementação padrão do escalonador padrão da ferramenta *Kubernetes* (*kube-scheduler*), quanto para evidenciar as possibilidades de implementação multi-objetivo utilizando a metodologia de rótulos.

O ecossistema utilizado na execução dos experimentos foi constituído a partir da construção de uma infraestrutura *multi-cloud* composta por nós computacionais de grandes fornecedores do mercado, sendo eles: *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)*, *Microsoft Azure* e a *Oracle Cloud Infrastructure (OCI)*, onde em cada um deles foram configuradas várias instâncias computacionais, definidas através de rótulos, para serem utilizadas em diferentes cenários. Além disso, para demonstrar de forma prática o comportamento do escalonador personalizado, foram selecionadas duas aplicações *open-source* de *benchmark* no consumo de recursos de *hardware*, como CPU (*vish/stress*[1]) e memória (*polinux/stress*[2]), a fim de obter resultados mais próximos às aplicações corporativas.

A análise dos resultados obtidos foi realizada utilizando métricas qualitativas, destacando a taxa de afinidade obtida ao término do processo de disponibilização das instâncias computacionais, comparando os requisitos de negócio definidos através de rótulos nas cargas de trabalho, com os nós computacionais que receberam tais instâncias. Também foi realizada uma análise de desempenho de ambos processos, comparando o tempo de escalonamento utilizando o orquestrador padrão e o personalizado.

A Seção 5.1 apresenta as configurações dos nós computacionais em seus respectivos fornecedores, bem como as cargas de trabalho a serem utilizadas, com suas marcações hipotéticas relacionadas a possíveis restrições de regras de negócio. Os cenários de testes construídos para execução dos experimentos e os objetivos a serem observados estão descritos na Seção 5.2. Na Seção 5.8 são apresentados os resultados dos experimentos realizados, detalhando os resultados qualitativos obtidos e o desempenho na execução dos processos, considerando o escalonador padrão e o personalizado.

5.1 Configuração do Ambiente de Teste

Com o objetivo de demonstrar a aplicação dos benefícios do escalonador personalizado foi construído um ecossistema em *cluster*, contendo vários nós computacionais

[1] <https://hub.docker.com/r/vish/stress>
[2] <https://hub.docker.com/r/polinux/stress>

disponibilizados através de diferentes fornecedores, constituindo assim um ambiente *multi-cloud*, apresentado na Tabela 5.

Tabela 5 – Nós computacionais participantes do *cluster* em ambiente *multi-cloud*.

Provedor	**Nó**	**Configuração**	**Especificações**	**Custo (R$/h)**
Azure	azure-worker-01	Standard_B1s	CPU=1 Memória=1gb	0,1086
Azure	azure-worker-02	Standard_B1s	CPU=1 Memória=1gb	0,1086
Azure	azure-worker-03	Standard_B1s	CPU=1 Memória=1gb	0,1086
Azure	azure-worker-04	Standard_B1s	CPU=1 Memória=1gb	0,1086
Azure	azure-worker-05	Standard_A1_v2	CPU=1 Memória=2gb	0,2911
AWS	aws-worker-01	t2.medium	CPU=2 Memória=4gb	0,2604
AWS	aws-worker-02	t2.medium	CPU=2 Memória=4gb	0,2604
AWS	aws-worker-03	t2.medium	CPU=2 Memória=4gb	0,2604
AWS	aws-worker-04	t2.medium	CPU=2 Memória=4gb	0,2604
AWS	aws-worker-05	t2.medium	CPU=2 Memória=4gb	0,2604
Google	gcp-worker-01	e2-micro	CPU=0.25 Memória=1gb	0,0470
Google	gcp-worker-02	e2-micro	CPU=0.25 Memória=1gb	0,0470
Google	gcp-worker-03	e2-micro	CPU=0.25 Memória=1gb	0,0470
Google	gcp-worker-04	e2-micro	CPU=0.25 Memória=1gb	0,0470
Google	gcp-worker-05	e2-micro	CPU=0.25 Memória=1gb	0,0470
Oracle	oracle-master-01	VM.Standard.E2.2	CPU=2 Memória=16gb	0,3164
Oracle	oracle-worker-01	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-02	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-03	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-04	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-05	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-06	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-07	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-08	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-09	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582
Oracle	oracle-worker-10	VM.Standard.E2.1	CPU=1 Memória=8gb	0,1582

Durante a instalação e configuração da ferramenta de orquestração *Kubernetes*, foi instituído um destes nós computacionais como sendo o *master*, representando na Tabela 5 pelo nó *oracle-master-01*, a partir disso, todos os processos de *schedule*, *re-schedule* e *auto-scale* serão administrador por este nó, direcionando os *pods* das cargas de trabalho de acordo com a definição realizada pelo escalonador.

Para demonstrar o comportamento do escalonador personalizado, foram atribuídos rótulos aos nós computacionais, de forma a destacar diferentes características quanto a sua localização geográfica, arquitetura de hardware e utilização de energias renováveis. As Tabelas 6, 7, 8 e 9 apresentam a configuração de rótulos atribuída a cada um dos nós computacionais que compõem o ambiente *multi-cloud*.

Tabela 6 – Rótulos definidos para os nós computacionais hospedados na nuvem pública *Oracle Cloud*, fornecida pela *Oracle*.

Provedor	**Nó**	**Rótulos**		
Oracle	oracle-master-01	cloud_vendor=oracle dc_country=usa dense_io=yes on_ramp=no	dc_continent=america dc_zone=us-phoenix-1 os_type=linux spot=no	dc_region=north green_energy_perc=11 app_environment=prod price=0,3164
Oracle	oracle-worker-01	cloud_vendor=oracle dc_country=usa dense_io=yes on_ramp=no	dc_continent=america dc_zone=us-phoenix-1 os_type=linux spot=no	dc_region=north green_energy_perc=11 app_environment=prod price=0,1582
Oracle	oracle-worker-02	cloud_vendor=oracle dc_country=usa dense_io=yes on_ramp=no	dc_continent=america dc_zone=us-phoenix-1 os_type=linux spot=no	dc_region=north green_energy_perc=11 app_environment=dev price=0,1582
Oracle	oracle-worker-03	cloud_vendor=oracle dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=us-phoenix-1 os_type=linux spot=no	dc_region=north green_energy_perc=11 app_environment=dev price=0,1582
Oracle	oracle-worker-04	cloud_vendor=oracle dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=us-sanjose-1 os_type=linux spot=no	dc_region=north green_energy_perc=100 app_environment=prod price=0,1582
Oracle	oracle-worker-05	cloud_vendor=oracle dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=us-sanjose-1 os_type=linux spot=no	dc_region=north green_energy_perc=100 app_environment=prod price=0,1582
Oracle	oracle-worker-06	cloud_vendor=oracle dc_country=brazil dense_io=no on_ramp=no	dc_continent=america dc_zone=sa-saopaulo-1 os_type=linux spot=no	dc_region=south green_energy_perc=88 app_environment=prod price=0,1582
Oracle	oracle-worker-07	cloud_vendor=oracle dc_country=brazil dense_io=yes on_ramp=no	dc_continent=america dc_zone=sa-saopaulo-1 os_type=linux spot=no	dc_region=south green_energy_perc=88 app_environment=prod price=0,1582
Oracle	oracle-worker-08	cloud_vendor=oracle dc_country=brazil dense_io=no on_ramp=no	dc_continent=america dc_zone=sa-saopaulo-1 os_type=linux spot=no	dc_region=south green_energy_perc=88 app_environment=prod price=0,1582
Oracle	oracle-worker-09	cloud_vendor=oracle dc_country=brazil dense_io=no on_ramp=yes	dc_continent=america dc_zone=equinix-sp4 os_type=linux spot=no	dc_region=south green_energy_perc=0 app_environment=prod price=0,1582
Oracle	oracle-worker-10	cloud_vendor=oracle dc_country=uk dense_io=yes on_ramp=no	dc_continent=europe dc_zone=uk-cardiff-1 os_type=windows spot=no	dc_region=central green_energy_perc=100 app_environment=dev price=0,1582

Tabela 7 – Rótulos definidos para os nós computacionais hospedados na nuvem pública *Azure*, fornecida pela *Microsoft*.

Provedor	**Nó**	**Rótulos**		
Azure	azure-worker-01	cloud_vendor=azure dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=us-west-2 os_type=windows spot=yes	dc_region=north green_energy_perc=80 app_environment=dev price=0,1086
Azure	azure-worker-02	cloud_vendor=azure dc_country=brazil dense_io=no on_ramp=no	dc_continent=america dc_zone=sa-east-1 os_type=windows spot=yes	dc_region=south green_energy_perc=50 app_environment=dev price=0,1086
Azure	azure-worker-03	cloud_vendor=azure dc_country=norway dense_io=yes on_ramp=yes	dc_continent=europe dc_zone=digiplex-oslo os_type=windows spot=yes	dc_region=north green_energy_perc=100 app_environment=dev price=0,1086
Azure	azure-worker-04	cloud_vendor=azure dc_country=japan dense_io=yes on_ramp=no	dc_continent=asia dc_zone=ap-northeast-1 os_type=linux spot=no	dc_region=northeast green_energy_perc=100 app_environment=dev price=0,1086
Azure	azure-worker-05	cloud_vendor=azure dc_country=australia dense_io=no on_ramp=no	dc_continent=oceania dc_zone=ap-southeast-2 os_type=linux spot=no	dc_region=south green_energy_perc=50 app_environment=prod price=0,2911

Tabela 8 – Rótulos definidos para os nós computacionais hospedados na nuvem pública *Amazon Web Services*, fornecida pela *Amazon*.

Provedor	**Nó**	**Rótulos**		
AWS	aws-worker-01	cloud_vendor=aws dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=eastus2 os_type=linux spot=yes	dc_region=north green_energy_perc=100 app_environment=dev price=0,2604
AWS	aws-worker-02	cloud_vendor=aws dc_country=brazil dense_io=yes on_ramp=no	dc_continent=america dc_zone=brazilsouth os_type=linux spot=yes	dc_region=south green_energy_perc=0 app_environment=dev price=0,2604
AWS	aws-worker-03	cloud_vendor=aws dc_country=brazil dense_io=no on_ramp=yes	dc_continent=america dc_zone=ascenty os_type=linux spot=yes	dc_region=south green_energy_perc=0 app_environment=dev price=0,2604
AWS	aws-worker-04	cloud_vendor=aws dc_country=chile dense_io=yes on_ramp=no	dc_continent=america dc_zone=chilecentral os_type=linux spot=yes	dc_region=south green_energy_perc=0 app_environment=dev price=0,2604
AWS	aws-worker-05	cloud_vendor=aws dc_country=france dense_io=no on_ramp=no	dc_continent=europe dc_zone=francecentral os_type=linux spot=yes	dc_region=central green_energy_perc=80 app_environment=dev price=0,2604

Tabela 9 – Rótulos definidos para os nós computacionais hospedados na nuvem pública *Google Cloud*, fornecida pela *Google*.

Provedor	**Nó**	**Rótulos**		
Google	gcp-worker-01	cloud_vendor=gcp dc_country=usa dense_io=no on_ramp=no	dc_continent=america dc_zone=us-west4 os_type=linux spot=yes	dc_region=north green_energy_perc=19 app_environment=dev price=0,0470
Google	gcp-worker-02	cloud_vendor=gcp dc_country=usa dense_io=no on_ramp=yes	dc_continent=america dc_zone=equinix-sv1 os_type=linux spot=yes	dc_region=north green_energy_perc=0 app_environment=dev price=0,0470
Google	gcp-worker-03	cloud_vendor=gcp dc_country=brazil dense_io=no on_ramp=no	dc_continent=america dc_zone=sa-east1 os_type=linux spot=no	dc_region=south green_energy_perc=88 app_environment=dev price=0,0470
Google	gcp-worker-04	cloud_vendor=gcp dc_country=belgium dense_io=no on_ramp=no	dc_continent=europe dc_zone=europe-west1 os_type=linux spot=no	dc_region=central green_energy_perc=79 app_environment=dev price=0,0470
Google	gcp-worker-05	cloud_vendor=gcp dc_country=taiwan dense_io=yes on_ramp=no	dc_continent=asia dc_zone=asia-east1 os_type=linux spot=no	dc_region=south green_energy_perc=18 app_environment=dev price=0,0470

A definição destes rótulos levou em consideração fontes públicas de acesso à informação, que possuem compartilhamento pelos próprios provedores de nuvem pública através de seus portais, onde apresentam detalhes relacionados às suas áreas territoriais de cobertura, modelo do fornecimento em relação à estrutura física, podendo ser própria ou em formato de sub-locação (*colocation* ou *on-ramp*) e custos computacionais tarifados por hora. Rótulos relacionados ao sistema operacional e tipo do armazenamento de dados (*dense_io*) são oriundos da escolha das configurações dos nós computacionais durante sua etapa de provisionamento. Já o rótulo referente ao ambiente de execução das aplicações (*app_environment*) houve a definição empírica, realizada para um melhor estudo de caso nos cenários de testes propostos na Seção 5.2, aplicando parâmetros comumente utilizados em ambientes corporativos. Por fim, o rótulo relacionado ao percentual de representativamente no consumo de energia verde pelos nós computacionais foi extraído de relatórios compartilhados pelos fornecedores *Amazon Web Services (AWS)*[3], *Google Cloud Platform (GCP)*[4], *Microsoft Azure*[5] e a *Oracle Cloud Infrastructure (OCI)*[6], onde cada um deles destacam o comprometimento em avançar na redução e neutralização da emissão de CO_2 resultantes de suas operações.

No Apêndice A foram organizadas informações técnicas relacionadas a configuração de rede para permitir à ferramenta de orquestração *Kubernetes* realizar a comunicação dos

[3] <https://sustainability.aboutamazon.com/about/around-the-globe>
[4] <https://cloud.google.com/sustainability/region-carbon#data>
[5] <https://azure.microsoft.com/en-us/global-infrastructure/sustainability>
[6] <https://www.oracle.com/corporate/citizenship/sustainability/clean-cloud.html>

contêineres através dos diferentes fornecedores de nuvem pública utilizados, uma vez que cada um deles possui regras de segurança específicas e componentes de gestão particulares.

5.2 Cenários de Testes

Inicialmente, foram definidos cenários com atribuição de características oriundas de requisitos de negócio, os quais definem o comportamento desejado pelo orquestrador durante o escalonamento das instâncias computacionais (*pods*), tais características tiveram suas definições representadas através de rótulos em suas respectivas cargas de trabalho. Para uma maior variação de resultados, foram construídos 5 cenários, cada um contendo algumas restrições definidas de acordo com o objetivo ao qual o mesmo se destinava.

O Experimento 1 apresenta um cenário simples, considerando uma aplicação única de usuário, que possui algumas características opcionais e restritivas quanto à escolha do nó computacional. No Experimento 2 o cenário é semelhante ao anterior, porém, neste foram consideradas múltiplas aplicações de usuário, com o objetivo de demonstrar a sua aplicação em cenários onde a administração do ambiente *multi-cloud* contempla cargas de trabalho de diferentes consumidores, e com suas respectivas regras restritivas.

Na sequência, são apresentados dois experimentos que objetivam a comparação do escalonador personalizado com escalonadores propostos no estado da arte, conforme apresentado no Capítulo 3.1. No Experimento 3, compara-se o *label-affinity-scheduler* com o escalonador proposto por (RODRIGUEZ; BUYYA, 2018), o qual apresenta-se um escalonador com foco na redução dos nós computacionais contratados em nuvens públicas, com a utilização da menor quantidade possível de *VMs*, utilizando toda a sua capacidade antes de realizar a alocação de *pods* em uma nova *VM* a ser contratada. No Experimento 4 utiliza-se um cenário em que a alocação das cargas de trabalho leva em consideração os nós computacionais localizados em regiões com a menor intensidade de emissão de gás carbônico. Este experimento está relacionado ao trabalho de (JAMES; SCHIEN, 2019), no qual dados foram obtidos de tabelas externas, e aplicado em processo de escalonamento dentro da nuvem *Azure*, utilizando nós computacionais em diferentes regiões do planeta. Por fim, o Experimento 5 realiza um teste de desempenho utilizando aplicações com diferentes tamanhos de instâncias, realizando um comparativo entre o *label-affinity-scheduler* e a implementação padrão da ferramenta de orquestração *Kubernetes* (*kube-scheduler*).

Em todos os experimentos as cargas de trabalho foram submetidas ao orquestrador padrão da ferramenta *Kubernetes* (*kube-scheduler*) e também ao escalonador personalizado, denominado de "*label-affinity-scheduler*". Ao término do processo de escalonamento, foi registrado o resultado final obtido da distribuição através dos nós computacionais disponíveis nas Tabelas 6, 7, 8 e 9, respeitando suas respectivas definições de rótulos. A Tabela 10

sintetiza os experimentos de acordo com suas características e objetivos propostos durante o processo de escalonamento.

Tabela 10 – Resumo dos experimentos realizados.

Experimento	**Características**	**Objetivos**
Experimento 1	Distribuição de 50 *pods* de uma única aplicação	Evidenciar a escolha dos nós computacionais em conformidade com a especificação de rótulos na carga de trabalho
Experimento 2	Distribuição de 150 *pods*, referente a três aplicações	Evidenciar a escolha dos nós computacionais em conformidade com a especificação de rótulos na carga de trabalho
Experimento 3	Definição de um ou mais rótulos na aplicação para que a escolha dos nós computacionais aconteça pela oferta mais econômica	Comparativo com solução proposta por (RODRIGUEZ; BUYYA, 2018)
Experimento 4	Definição de um ou mais rótulos na aplicação para que a escolha dos nós computacionais aconteça de acordo com os níveis de uso de energia verde	Comparativo com solução proposta por (JAMES; SCHIEN, 2019)
Experimento 5	Escalonamento de 4 aplicações com diferentes quantidades de *pods*	Análise de desempenho do escalonador personalizado, comparado à implementação padrão

Os experimentos para validação do escalonador consideram cenários hipotéticos em que, durante o levantamento de requisitos da aplicação, definiu-se algumas características técnicas para eleição do nó computacional de execução do *pod*. Logo, qualquer *pod* que não respeite tais regras, representa um não atingimento deste requisito. Os impactos resultantes dessas infrações podem representar gastos financeiros acima do planejado, riscos a acordos comerciais e legais (atendimento de questões relativas à Lei Geral de Proteção de Dados (LGPD), por exemplo) ou não concordância com políticas ambientais difundidas pela empresa contratante dos serviços.

5.3 Experimento 1

5.3.1 Configurações do cenário

O experimento considerou a execução da carga de trabalho App A, que requisita 50 instâncias (*pods*) da carga de trabalho "polinux/stress". Na Tabela 11 estão definidos os rótulos a serem considerados durante o processo de escalonamento, os quais definem as características desejáveis ou obrigatórias durante o processo de análise dos nós computacionais disponíveis.

Tabela 11 – Definição de rótulos para as cargas de trabalho empregadas no cenário de teste.

Carga de Trabalho	**Rótulos**
App A (50 *pods*) polinux/stress	(1) dense_io=eq_-yes.5 (2) os_type=eq-windows (3) app_environment=eq-dev (4) on_ramp=eq-no

5.3.2 Resultados

A Figura 14 apresenta o resultado do escalonador padrão do *Kubernetes* (*kube-scheduler*) para alocar os *pods* da carga de trabalho descrita na Tabela 11, bem como o resultado do *label-affinity-scheduler*.

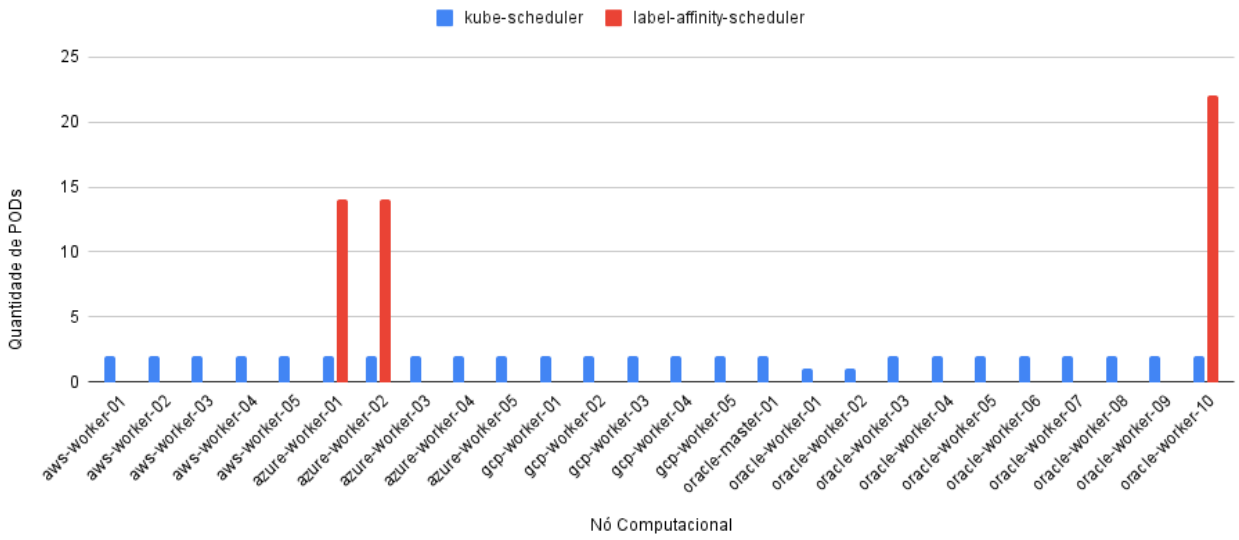

Figura 14 – Distribuição dos *pods* utilizando escalonador padrão (A) e personalizado (B).

Como comportamento padrão, o escalonador *kube-scheduler* considera todos os nós computacionais ativos no *cluster* orquestrado como elegíveis de alocação dos *pods*, logo, obtém-se o resultado visualizado na Figura 14, em que há um equilíbrio na alocação através de todos os nós computacionais.

Já na abordagem do escalonador personalizado, são considerados os rótulos para garantir a afinidade entre os *pods* e o nó computacional, o resultado final leva em consideração a disponibilidade de nós com as características definidas como requisito para cada aplicação. Visualizando a Figura 14, percebe-se que há uma diferença na alocação de *pods* entre os nós computacionais, isto deve-se ao fato de haver uma definição para priorizar aqueles com rótulo "*dense_io*".

Ao realizar uma análise qualitativa em relação ao resultado final, tem-se para o cenário da aplicação A utilizando o escalonador padrão, uma taxa de 12% de sucesso

na alocação dos *pods* a nós que contenham rótulos desejáveis, enquanto no escalonador personalizado atinge 100% de sucesso. Note que, a taxa de sucesso do escalonador padrão pode ser maior ou menor, uma vez que, dependendo dos nós disponíveis e da ordem de alocação, alguns dos requisitos podem ser consequentemente atendidos.

5.4 Experimento 2

5.4.1 Configurações do cenário

O Experimento 2 tem o objetivo de apresentar a aplicação do escalonador personalizado em um cenário multi-usuário, no qual são atendidos diferentes requisitos de negócio originados por diferentes características de cada aplicação. Nele, foram consideradas a execução de três cargas de trabalho, denominadas "App B", "App C" e "App D", onde cada uma delas requisita 50 instâncias (*pods*) da carga de trabalho "vish/stress". Na Tabela 12 estão definidos os rótulos a serem considerados durante o processo de escalonamento, eles definem as características desejáveis ou obrigatórias durante o processo de análise dos nós computacionais disponíveis.

Tabela 12 – Definição de rótulos para as cargas de trabalho empregadas no cenário de teste.

Carga de Trabalho	**Rótulos**	
App B (50 *pods*) vish/stress	(1) price=le-0.15	(2) cloud_vendor=in-azure-gcp
App C (50 *pods*) vish/stress	(1) spot=eq_-yes (3) dc_country=eq-usa	(2) os_type=eq-linux
App D (50 *pods*) vish/stress	(1) on_ramp=eq-no (3) dc_country=eq_-brazil.5	(2) app_environment=eq-prod

5.4.2 Resultados

A requisição de todas instâncias das cargas de trabalho "App B", "App C" e "App D" foi realizada de forma simultânea, em um primeiro momento não tendo nenhuma definição de escalonador personalizado definido em seu arquivo de configuração, indicando à ferramenta de orquestração *Kubernetes* para utilizar seu próprio componente de escalonamento (*kube-scheduler*).

Depois disso, foi adicionada a referência ao escalonador personalizado (*label-affinity-scheduler*), e novamente realizado o escalonamento das cargas de trabalho. Neste momento houve a análise restritiva e desejável de afinidade de rótulos entre as cargas de trabalho e

todos os nós computacionais participantes do *cluster*, a fim de garantir que tais condições fossem satisfeitas.

As Figuras 15, 16 e 17, apresentam os resultados obtidos do processo de escalonamento da cargas de trabalho "App B", "App C" e "App D", respectivamente, utilizando os escalonadores *kube-scheduler* e *label-affinity-scheduler*. Nesta visualização destaca-se o fato do escalonador *kube-scheduler* ter realizado uma distribuição *round-robin* das instâncias da aplicação (*pods*), não considerando fatores como a capacidade computacional ociosa, a quantidade de *pods* já alocados referente a outras cargas de trabalho preexistentes, ou qualquer requisito de negócio da aplicação.

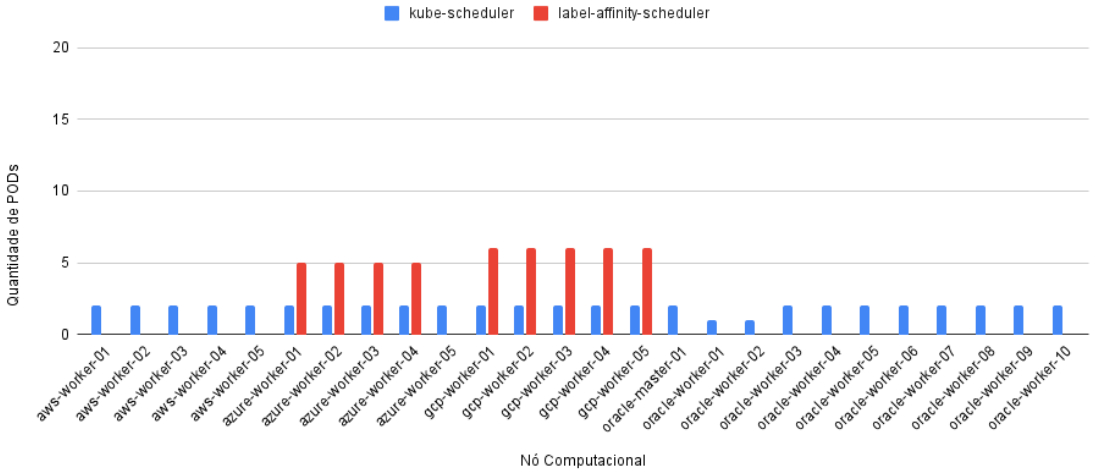

Figura 15 – Distribuição dos *pods* da carga de trabalho "App B" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

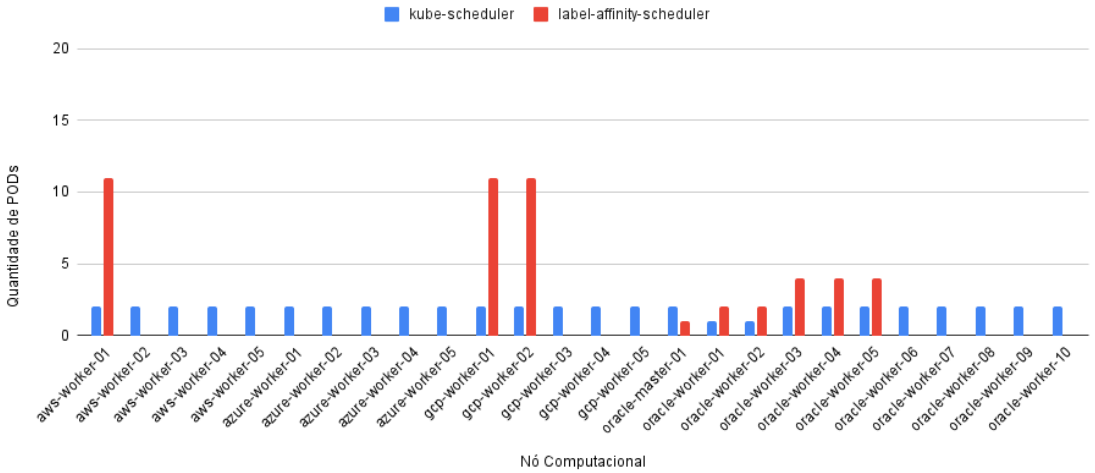

Figura 16 – Distribuição dos *pods* da carga de trabalho "App C" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

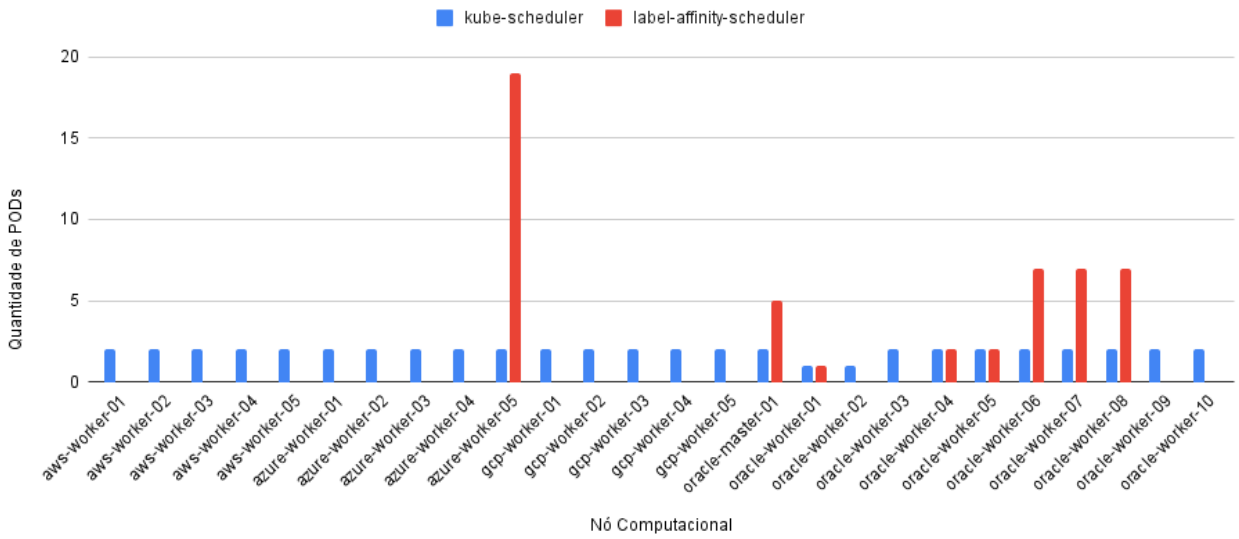

Figura 17 – Distribuição dos *pods* da carga de trabalho "App D" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

Como resultado do escalonamento utilizando o *label-affinity-scheduler* tivemos comportamentos distintos de distribuição dos *pods*, devido a requisitos distintos para cada carga de trabalho.

Para a carga de trabalho referente a "App B", a concentração dos *pods* em alguns dos nós computacionais acontece devido ao rótulo restritivo relacionado aos custos da instância computacional, especificado na Tabela 12, onde qualquer nó computacional que tenha valor acima do definido seja descartado.

Em relação a carga de trabalho "App C", houve a definição de um rótulo desejável, relacionado à instância computacional ter a característica "*spot*" - que possuem um baixo custo financeiro por utilizarem a capacidade ociosa das nuvens computacionais - com isso, algumas instâncias computacionais tiveram uma maior concentração de *pods*, porém, estas mesmas instâncias não tiveram exclusividade, pois durante o escalonamento há um classificador de prioridade da alocação que considera a capacidade computacional ociosa, e isto, aliado ao fato deste rótulo ser do tipo desejável, fez com que houvesse a distribuição através de outros nós computacionais.

Por fim, a carga de trabalho "App D" teve um comportamento semelhante ao observado na alocação das instâncias da carga de trabalho "App C", em que houve uma definição desejável de alocação em nós computacionais com o rótulo *dc_country=brazil*, porém, com uma característica extra relacionada à importância deste rótulo, onde tais nós computacionais possuem um "bônus" de prioridade, fazendo com que a concentração de *pods* seja maior.

A Figura 18 apresenta o compilado de todas as instâncias das aplicações “App B”, “App C” e “App D”, que foram distribuídas através de todos os nós computacionais, utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

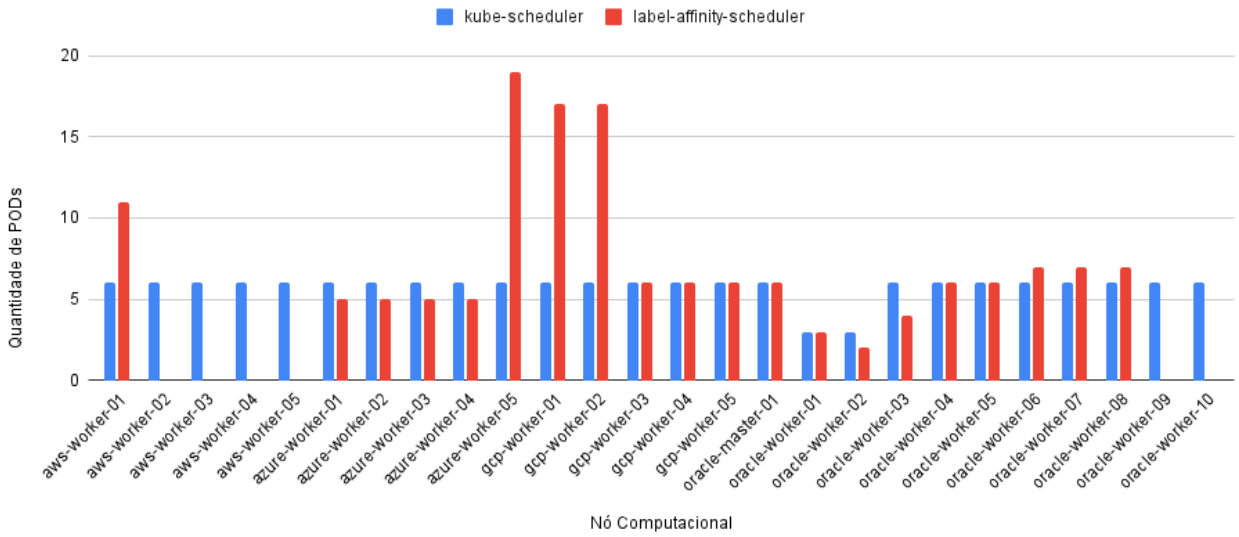

Figura 18 – Distribuição de todas as instâncias das cargas de trabalho “App B”, “App C” e “App D”.

É possível notar na Figura 18 uma concentração de *pods* realizada nos nós computacionais *azure-worker-05*, *gcp-worker-01* e *gcp-worker-02*, isto é resultado da afinidade existente entre estes três nós com os rótulos definidos nas cargas de trabalho “App B”, “App C” e “App D”. Considerando um cenário hipotético, onde a quantidade de *pods* para estas cargas de trabalho fosse superior, ocasionando uma concentração nestes mesmos nós computacionais que exceda o limite comportado pela ferramenta de orquestração *Kubernetes*, que é de 110 *pods*, a aplicação ficaria em um estado de pendência, já que a quantidade desejada de instâncias não foi atingida. Como há restrição por rótulos quanto a compatibilidade dos nós computacionais, este cenário remete a necessidade de contratação de novos nós que satisfaçam os requisitos, ou a uma redução da quantidade de *pods* definida na carga de trabalho.

5.5 Experimento 3

5.5.1 Configurações do cenário

Com o objetivo de analisar a aplicabilidade do escalonador *label-affinity-scheduler* em implementações já abordadas pelo estado-da-arte, este experimento tenta replicar o cenário implementado por (RODRIGUEZ; BUYYA, 2018), no qual objetiva-se alcançar maior eficiência dos custos financeiros que envolvem a disponibilização de cargas de trabalho

em um ambiente de computação em nuvem, não se limitando apenas a disponibilização inicial, mas também durante as fases de escalonamento incremental e durante eventos de re-escalonamento de instâncias da aplicação durante eventos de indisponibilidade de algum nó que esteja executando alguma destas instâncias.

Em (RODRIGUEZ; BUYYA, 2018), foi utilizada uma estratégia de nuvem computacional de fornecedor único, composta por dois nós computacionais. A partir disso, foi construído um escalonador personalizado utilizando um algoritmo *best-fit bin-packing*, que realiza a interceptação de todas os *pods* da carga de trabalho, e estes por sua vez vão sendo direcionados a um nó computacional que tenha a menor quantidade de recursos físicos disponíveis, porém, que tenha capacidade de acomodá-lo. Esta estratégia faz com que a menor quantidade de nós computacionais possível seja necessária para atender a demanda da carga de trabalho, e consequentemente, possibilitando que os nós ociosos possam ser desativados e reduzir o custo operacional por hora.

Considerando os recursos de rótulos implementados pelo escalonador *label-affinity-scheduler*, é possível definir marcação de preço máximo aceitável, ao ponto que, quanto menor o valor encontrado na definição deste rótulo no nó computacional, maior a pontuação de afinidade o mesmo receberá, concentrando a alocação dos *pods* naqueles de menor custo. Em cenários de empate do valor definido para o rótulo em dois ou mais nós computacionais, a prioridade sempre ficará com aquele que possui a maior quantidade de *pods* em execução, e que tenha recursos físicos suficientes para execução da instância.

O comportamento resultante deste processo será o direcionamento de instâncias da aplicação até que seus recursos de *hardware* (CPU e memória) sejam totalmente consumidos, atingindo assim o objetivo de forma semelhante ao proposto no trabalho de (RODRIGUEZ; BUYYA, 2018).

A Tabela 13 apresenta um modelo de carga de trabalho submetida ao escalonador *label-affinity-scheduler*, tendo uma definição de rótulo para considerar o atributo *price* com um valor máximo aceitável. Para uma melhor comparação posterior, foi definida a quantidade de 50 instâncias da aplicação, assim como utilizado por (RODRIGUEZ; BUYYA, 2018). Para uma análise quantitativa da representatividade financeira obtida por este modelo, o mesmo também será realizado utilizando o escalonador padrão *kube-scheduler*.

Tabela 13 – Definição de rótulos para a carga de trabalho empregada no cenário de teste.

Carga de Trabalho	**Rótulos**
App E (50 *pods*) polinux/stress	(1) price=le-0.10

Como o rótulo "*price*" definido na carga de trabalho "App E" possui o operador lógico *le* (menor ou igual), o algoritmo implementado pelo escalonador *label-affinity-*

scheduler dará uma maior pontuação aos nós computacionais que possuírem o menor valor neste rótulo durante a etapa de prioridade, garantindo desta forma que aqueles com o menor custo operacional tenham preferência sobre os demais.

5.5.2 Resultados

Assim como nos demais experimentos, a requisição de todas as instâncias da carga de trabalho "App E" foi realizada em duas etapas. Na primeira utilizando o escalonador padrão, *kube-scheduler*, já na segunda, foi definido o escalonador personalizado *label-affinity-scheduler*.

Na Figura 19 é possível visualizar a diferença de distribuição de todos os *pods*, entre os escalonadores *kube-scheduler* e *label-affinity-scheduler*. Como o objetivo era priorizar aqueles nós computacionais com o menor custo financeiro, percebe-se que há um grande volume computacional ocioso em cada um destes nós, uma vez que estão sendo subutilizados.

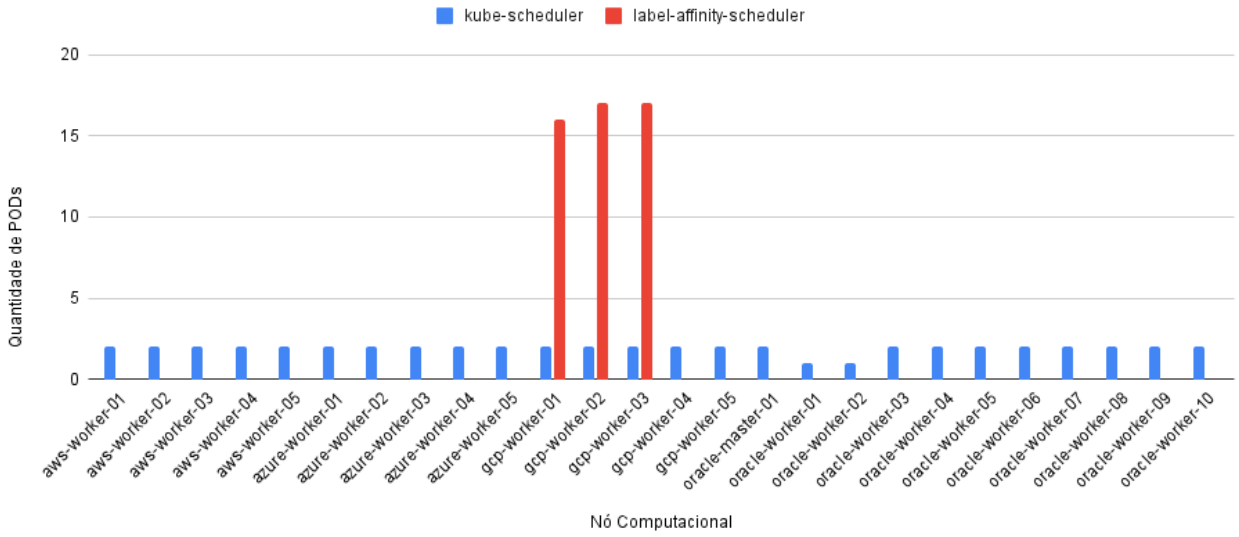

Figura 19 – Distribuição de todas as instâncias da carga de trabalho "App E".

Da mesma forma como o trabalho de (RODRIGUEZ; BUYYA, 2018) propõe, os *pods* são recebidos de forma contínua e direcionados ao nó computacional como a menor quantidade de recurso computacional disponível, e que satisfaça o mínimo exigido pelo mesmo. A Figura 20 apresenta o progresso da distribuição de cada um destes *pods* através dos nós computacionais que satisfizeram a restrição de custo definidas na Tabela 13 da carga de trabalho "App E", e também, garantindo que a menor quantidade de nós computacionais fosse utilizada.

Este comportamento permite que os nós computacionais não utilizados sejam desativados durante o período em que não houver necessidade de mais recursos computacionais

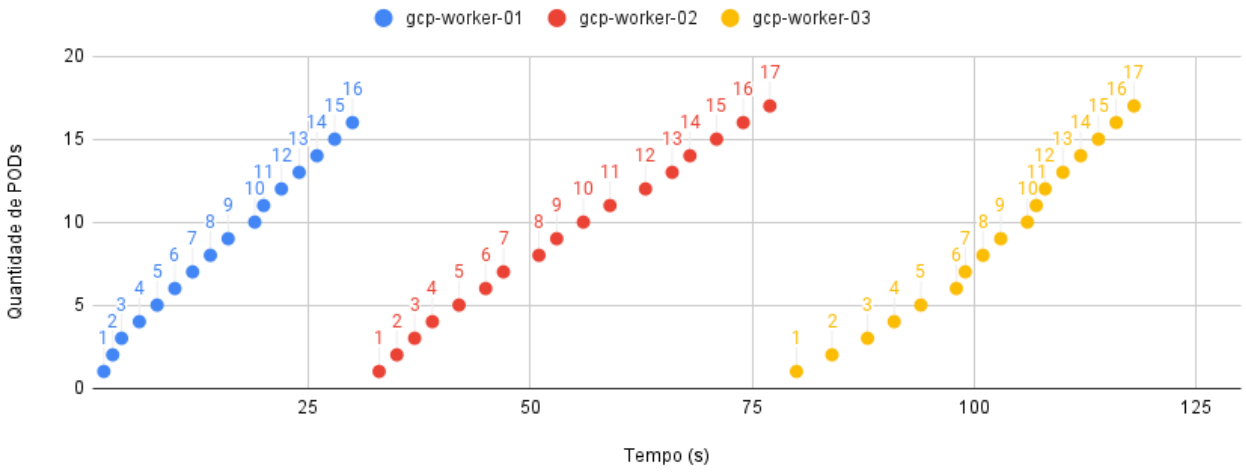

Figura 20 – Progresso da distribuição de todas as instâncias da carga de trabalho "App E" ao longo do tempo.

no ecossistema orquestrado, e novamente ativados (ou contratados) conforme o aumento da demanda.

Aplicar esta estratégia de alocação de *pods* é muito eficiente em termos de custo, uma vez que o modelo de tarifação deste tipo de serviço ocorre pela disponibilização do recurso, e não pelo consumo do mesmo, logo, quanto mais próximo da sua capacidade total de processamento for utilizada, melhor será a eficiência do investimento realizado.

Os resultados obtidos por (RODRIGUEZ; BUYYA, 2018) apresentam uma redução de 58% ao comparar o comportamento do escalonador personalizado proposto, com a implementação padrão do *Kubernetes*. Realizando um comparativo no mesmo formato, a Tabela 14 apresenta o custo total por hora para cada um dos nós computacionais, multiplicado pelo total de *pods* atribuído para cada um deles, nos dois cenários de utilização dos escalonadores *kube-scheduler* e *label-affinity-scheduler*.

Considerando o contexto atual, composto pelas configurações e precificação dos nós computacionais dentro de cada fornecedor, a economia foi de 3,4x. A diferença nos percentuais de economia obtidos observada entre os dois escalonadores personalizados sofre influência de vários fatores, como a quantidade de nós computacionais disponíveis no *cluster*, a política de preços de cada um deles e a quantidade de recursos físicos necessários para execução de cada instância da carga de trabalho.

Tabela 14 – Distribuição dos *pods* através dos nós computacionais e custo final empregando os escalonadores *kube-scheduler* e *label-affinity-scheduler*.

Provedor	**Nó**	**Custo (R$/h)**	***kube-scheduler* (*pods*)**	***label-affinity-scheduler* (*pods*)**
Azure	azure-worker-01	0,1086	2 (0,2172/h)	0 (0,0/h)
Azure	azure-worker-02	0,1086	2 (0,2172/h)	0 (0,0/h)
Azure	azure-worker-03	0,1086	2 (0,2172/h)	0 (0,0/h)
Azure	azure-worker-04	0,1086	2 (0,2172/h)	0 (0,0/h)
Azure	azure-worker-05	0,2911	2 (0,5822/h)	0 (0,0/h)
AWS	aws-worker-01	0,2604	2 (0,5208/h)	0 (0,0/h)
AWS	aws-worker-02	0,2604	2 (0,5208/h)	0 (0,0/h)
AWS	aws-worker-03	0,2604	2 (0,5208/h)	0 (0,0/h)
AWS	aws-worker-04	0,2604	2 (0,5208/h)	0 (0,0/h)
AWS	aws-worker-05	0,2604	2 (0,5208/h)	0 (0,0/h)
Google	gcp-worker-01	0,0470	2 (0,0940/h)	16 (0,752/h)
Google	gcp-worker-02	0,0470	2 (0,0940/h)	17 (0,799/h)
Google	gcp-worker-03	0,0470	2 (0,0940/h)	17 (0,799/h)
Google	gcp-worker-04	0,0470	2 (0,0940/h)	0 (0,0/h)
Google	gcp-worker-05	0,0470	2 (0,0940/h)	0 (0,0/h)
Oracle	oracle-master-01	0,3164	2 (0,6328/h)	0 (0,0/h)
Oracle	oracle-worker-01	0,1582	1 (0,1582/h)	0 (0,0/h)
Oracle	oracle-worker-02	0,1582	1 (0,1582/h)	0 (0,0/h)
Oracle	oracle-worker-03	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-04	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-05	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-06	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-07	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-08	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-09	0,1582	2 (0,3164/h)	0 (0,0/h)
Oracle	oracle-worker-10	0,1582	2 (0,3164/h)	0 (0,0/h)
			50 (8,0054/h)	**50 (2,35/h)**

5.6 Experimento 4

5.6.1 Configurações do cenário

Este experimento visa aplicar o escalonador *label-affinity-scheduler* no cenário implementado por (JAMES; SCHIEN, 2019), o qual envolveu a implementação de um escalonador para a ferramenta de orquestração *Kubernetes* para exploração e priorização de nós computacionais com a menor emissão de gás CO_2, denominado *low carbon kubernetes scheduler*. Fazendo parte da estratégia de escalonamento, não apenas a disponibilização inicial das instâncias da aplicação (*pods*), mas também realizando a revisão periódica e posterior migração para regiões com a menor intensidade de emissão do gás CO_2.

Para atingir o objetivo, (JAMES; SCHIEN, 2019) utilizaram uma *API* de dados climáticos de terceiros, chamada *Weatherbit.io*[7], para coleta de informações das regiões geográficas em que os *datacenters* que hospedam os nós computacionais estão localizados. Dentre os dados recebidos, estão informações referente ao nível de irradiação solar, tempe-

[7] <https://www.weatherbit.io>

ratura do ar e velocidade do vento. Este conjunto de parâmetros permitiu construir um algoritmo denominado *heliotropic*, responsável por identificar os *datacenters* com potencial menor índice de emissão de gás CO_2.

Como o ecossistema alvo dos experimentos é composto por diferentes fornecedores e cada um deles utiliza uma nomenclatura diferente para suas métricas de adesão a fontes energéticas livre da emissão de carbono optou-se por definir um rótulo denominado "*green_energy_perc*", onde em sua definição, um número de 100 representa a não incidência de emissão de gás CO_2. Migrar o consumo energético para o advento "verde" não é um conceito novo, as empresas têm se dedicado cada vez mais para alcançá-lo, seja através da adoção de estratégias na dissipação térmica dos equipamentos de *hardware*, virtualização, fontes energéticas naturais, dentre outras, culminando na redução ou mitigação das emissões de gás carbônico, (S; CHOLLI, 2021).

Assim como no trabalho de (JAMES; SCHIEN, 2019), utilizou-se como carga de trabalho alvo para execução das simulações foi utilizado a estrutura de projeto chamada BOINC[8] (*Berkeley Open Infrastructure for Network Computing*), em que a capacidade computacional é utilizada para processar pesquisas científicas, através de um *grid* computacional constituído por voluntários.

Utilizando os recursos de rótulo do escalonador *label-affinity-scheduler*, e as marcações previamente realizadas nos nós computacionais em relação aos seus percentuais de utilização de energia verde (*green_energy_perc*), conforme dados oficiais divulgados pelos fornecedores das nuvens públicas contratadas e apresentados na Seção 5.1, é possível definir um rótulo restritivo, de nível mínimo exigido em relação à utilização de energia verde, concentrando a alocação dos *pods* em nós computacionais com o maior valor para este rótulo, e que tenham recursos de *hardware* (CPU e memória) livres para alocação novos *pods*.

A Tabela 15 apresenta um modelo de carga de trabalho utilizando como imagem a mesma aplicação proposta por (JAMES; SCHIEN, 2019), porém, utilizando uma distribuição obtida do repositório do usuário *zilman*[9] na plataforma *Docker HUB*. Além disso, foi especificado um volume de 10 instâncias para execução, pois como se trata de uma aplicação de consumo intensivo de recursos de *hardware*, cada um dos nós computacionais não receberão mais de 1 *pod*.

Tabela 15 – Definição de rótulos para a carga de trabalho empregada no cenário de teste.

Carga de Trabalho	**Rótulos**
App F (10 *pods*) zilman/boinc	(1) green_energy_perc=gt-0

[8] <https://boinc.berkeley.edu>
[9] <https://hub.docker.com/r/zilman/boinc>

5.6.2 Resultados

O processo de escalonamento da carga de trabalho "App F" foi realizado em duas etapas, na primeira delas, identificando a afinidade e o cumprimento das restrições de rótulos, neste cenário, considerando a obrigatoriedade da informação referente a "*green_energy_perc*" ser maior que 0, já na segunda, foi realizada a etapa de priorização dos nós computacionais, onde é levado em consideração suas capacidades de *hardware*, a disponibilidade livre para alocação da quantidade requisitada pelo *pod*.

A Tabela 16 apresenta a lista de nós computacionais com suas respectivas pontuações referente à prioridade, obtida através do cálculo realizado pelo algoritmo do escalonador personalizado *label-affinity-scheduler*, e por se tratar de uma operação matemática (compreendida pelos operadores *gt, ge, lt, e le*), garante maior pontuação de acordo com a definição do nó computacional, além disso, aplica outro critério baseado na capacidade computacional ociosa de cada um deles.

Tabela 16 – Cálculo de prioridade dos nós computacionais considerando a definição de rótulos.

Provedor	**Nó**	**Prioridade**
Azure	azure-worker-01	177.753736
Azure	azure-worker-02	147.753736
Azure	azure-worker-03	197.753736
Azure	azure-worker-04	197.753736
Azure	azure-worker-05	147.753736
AWS	aws-worker-01	197.753736
AWS	aws-worker-02	0.000000
AWS	aws-worker-03	0.000000
AWS	aws-worker-04	0.000000
AWS	aws-worker-05	177.753736
Google	gcp-worker-01	115.119670
Google	gcp-worker-02	0.000000
Google	gcp-worker-03	185.753736
Google	gcp-worker-04	176.753736
Google	gcp-worker-05	115.753736
Oracle	oracle-master-01	87.774659
Oracle	oracle-worker-01	98.646146
Oracle	oracle-worker-02	98.646146
Oracle	oracle-worker-03	104.456005
Oracle	oracle-worker-04	193.456004
Oracle	oracle-worker-05	193.456005
Oracle	oracle-worker-06	181.456004
Oracle	oracle-worker-07	181.456005
Oracle	oracle-worker-08	181.456005
Oracle	oracle-worker-09	0.000000
Oracle	oracle-worker-10	193.456005

Outra característica a ser observada na Tabela 16 é relacionada aos nós computacionais que obtiveram prioridade "0.000000", sendo estes descartados no processo de escalonamento por não atenderem o rótulo definido na carga de trabalho, e o fato do mesmo ser obrigatório.

A Figura 21 exibe a evolução da disponibilização de cada *pod* da carga de trabalho, respeitando a priorização de calculada de acordo com a Tabela 16. Neste ponto, é possível visualizar que, a alocação de *pods* prioriza os nós computacionais com maior prioridade definida pelo escalonador *label-affinity-scheduler*.

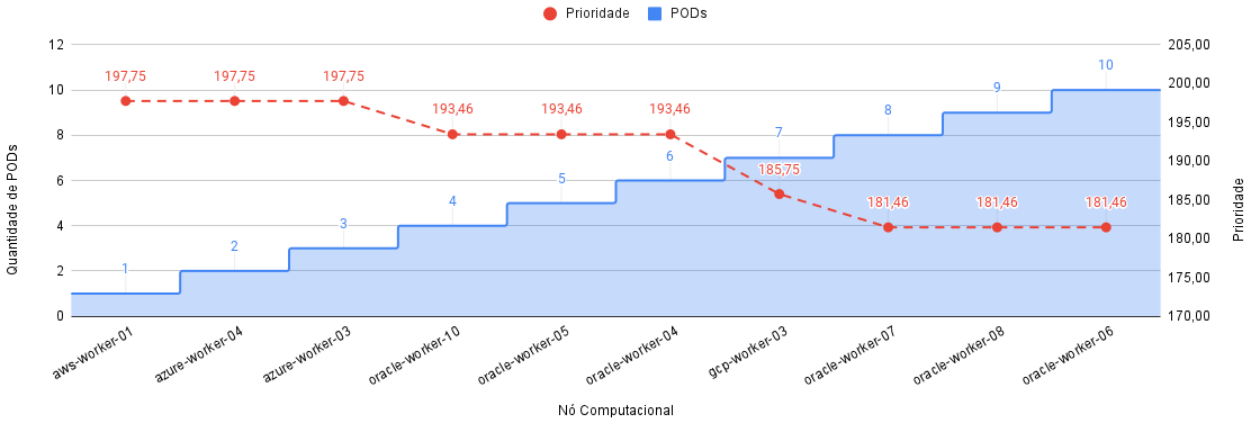

Figura 21 – Evolução da disponibilização dos *pods* da carga de trabalho "App F" de acordo com a prioridade de cada nó computacional.

O comparativo entre os dois escalonadores personalizados apresentou algumas diferenças em relação ao modelo de execução dos testes. No trabalho de (JAMES; SCHIEN, 2019) a execução da carga de trabalho envolvia um único *pod*, que passava por eventos de re-escalonamento conforme a horário do dia avançava e a *API* de dados climáticos *Weatherbit.io* indicava uma região geográfica com maiores níveis de irradiação solar. Já em nosso cenário, utilizando o escalonador *label-affinity-scheduler*, a definição de emissão de gás CO_2 não se altera, então optou-se por realizar o escalonamento de 10 *pods* para demonstrar a priorização dos nós computacionais de acordo com a definição quanto ao uso de energia verde.

Este cenário demonstrou a capacidade de aplicação *label-affinity-scheduler* à proposta do *low carbon kubernetes scheduler*, tendo como principal diferença a necessidade de definição prévia de rótulos, indicando os índices de utilização de energia verde, ao invés de extraí-los dinamicamente de uma *API*.

De certa forma, podem existir divergências conceituais na estratégia do escalonador proposto por (JAMES; SCHIEN, 2019), já que adotar índices de irradiação solar, temperatura do ar e velocidade do vento como parâmetros para calcular o índice de emissão de

gás CO_2 de *datacenters* pode não representar a realidade praticada pelos mesmos, já que outras fontes de energia e técnicas de resfriamento podem ser adotadas para minimizar o consumo energético das estruturas.

Fornecedores de nuvens computacionais públicas têm expandido suas ações e estratégias em busca de fornecer um serviço livre da emissão de gases poluentes, adotando meios alternativos de consumo energético e novas tecnologias que auxiliem neste processo, inclusive, anunciando publicamente este compromisso e prazos para cumprimento de metas. Alguns destes fornecedores com representatividade notória no segmento destacam tais ações em portais dedicados, como é o caso da *Amazon (AWS)*[10], *Google (GCP)*[11], *Microsoft (Azure)*[12] e a *Oracle (OCI)*[13].

Logo, adotar como variável os dados públicos compartilhados pelos próprios fornecedores em relação a sua emissão de gás CO_2 corrobora a favor do escalonador *label-affinity-scheduler* e seu escalonamento considerando afinidade por utilização de energia verde. Outro fator necessário para o sucesso da estratégia utilizada pelo escalonador *low carbon kubernetes scheduler*, é a necessidade prévia da contratação de serviços de computação em nuvem espalhadas por diferentes regiões de todo o planeta, garantindo a eficiência do processo de migração das instâncias das aplicações para tais regiões. O resultado de sua eficiência energética envolve a mensuração da emissão de gás CO_2 considerando os períodos em que as instâncias da aplicação estiveram em execução em cada nó computacional, durante diferentes períodos do dia. Diferente ao obtido pelo escalonador *label-affinity-scheduler*, em que não há migração de nó computacional após a disponibilização inicial.

Por fim, estender as funcionalidades encontradas no escalonador *low carbon kubernetes scheduler* para o *label-affinity-scheduler* envolveria a implementação de duas características, a primeira delas é implementar a integração com a *API* de dados climáticos *Weatherbit.io* para coleta de informações climáticas regionais, e a outra, seria a criação de um agendador de tarefas à ser executado periodicamente, de tal forma que ao identificar uma mudanças nos dados climáticos de uma região, executaria um processo de "exclusão" do *pod* localizado em um nó computacional menos indicado em termos energéticos. Fazendo com que a ferramenta de orquestração *Kubernetes* disponibilize um novo *pod* para escalonamento, e este por sua vez sendo direcionado a um novo nó computacional, com melhores índices de eficiência energética.

[10] <https://sustainability.aboutamazon.com>
[11] <https://cloud.google.com/sustainability>
[12] <https://www.microsoft.com/en-us/sustainability/azure>
[13] <https://www.oracle.com/solutions/green/cloud-operations.html>

5.7 Experimento 5

5.7.1 Configurações do cenário

Considerando que o escalonador personalizado *label-affinity-scheduler* implementa funcionalidades não compreendidas pelo escalonador *kube-scheduler*, como a análise de afinidade dos rótulos definidos para as cargas de trabalho com as definições dos nós computacionais, e como critério de desempate e balanceamento de recursos, a cada novo *pod* recebido para disponibilização, é verificado a capacidade ociosa de cada um destes nós.

Estas funcionalidades combinadas resultam em uma queda de desempenho para completar o processo de escalonamento, e para melhor compreensão do nível deste impacto foram construídos 4 cenários com as mesmas características de rótulos, com a diferença de que cada um deles possui diferentes quantidades de *pods* a serem escalonados, possibilitando assim uma melhor compreensão do quanto tais funcionalidades personalizadas impactam no resultado final do processo. A Tabela 17 apresenta os cenários considerados e seus parâmetros, e como o objetivo é monitorar o desempenho da distribuição de cada um dos *pods* aos nós computacionais, foi utilizada uma imagem de aplicação que apenas aguardando algum sinal de teclado, obtida do repositório do usuário *hendrikmaus*[14].

Tabela 17 – Definição de rótulos para as cargas de trabalho empregadas no cenário de teste.

Carga de Trabalho	**Rótulos**
App G (10 *pods*) hendrikmaus/kubernetes-dummy-image	(1) cloud_vendor=in-azure-aws-gcp-oracle (2) os_type=in-linux-windows (3) app_environment=notin-tst
App H (100 *pods*) hendrikmaus/kubernetes-dummy-image	(1) cloud_vendor=in-azure-aws-gcp-oracle (2) os_type=in-linux-windows (3) app_environment=notin-tst
App I (1000 *pods*) hendrikmaus/kubernetes-dummy-image	(1) cloud_vendor=in-azure-aws-gcp-oracle (2) os_type=in-linux-windows (3) app_environment=notin-tst
App J (2000 *pods*) hendrikmaus/kubernetes-dummy-image	(1) cloud_vendor=in-azure-aws-gcp-oracle (2) os_type=in-linux-windows (3) app_environment=notin-tst

[14] <https://hub.docker.com/r/hendrikmaus/kubernetes-dummy-image>

5.7.2 Resultados

O escalonamento de todas as instâncias das cargas de trabalho "App G", "App H", "App I" e "App J" foi realizado de forma individual e em dois momentos distintos, primeiro, utilizando o escalonador padrão do *Kubernetes* (*kube-scheduler*), e no segundo momento, com a definição do escalonador personalizado *label-affinity-scheduler*.

Com o objetivo de entender o impacto em diferentes tamanhos de aplicações e quantidades de instâncias necessárias, as Figuras 22, 23, 24 25 apresentam os resultados obtidos no processo de escalonamento, respectivamente, utilizando os escalonadores *kube-scheduler* e *label-affinity-scheduler*.

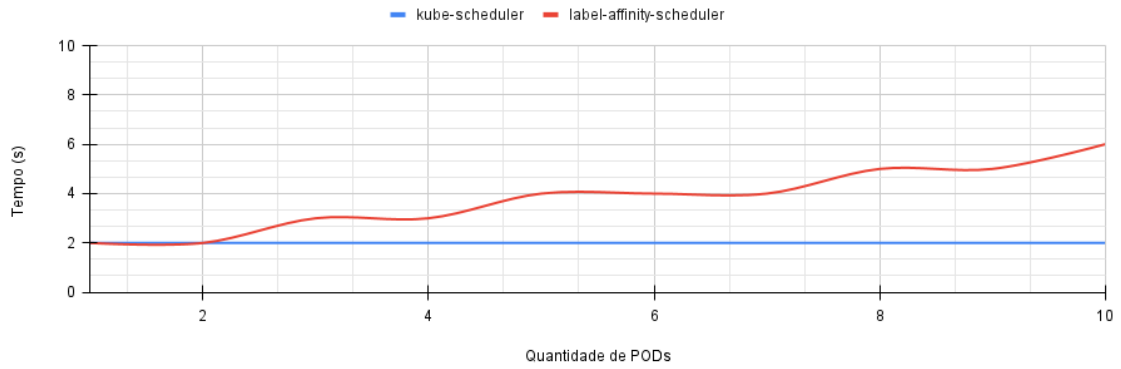

Figura 22 – Desempenho da distribuição dos *pods* da carga de trabalho "App G" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

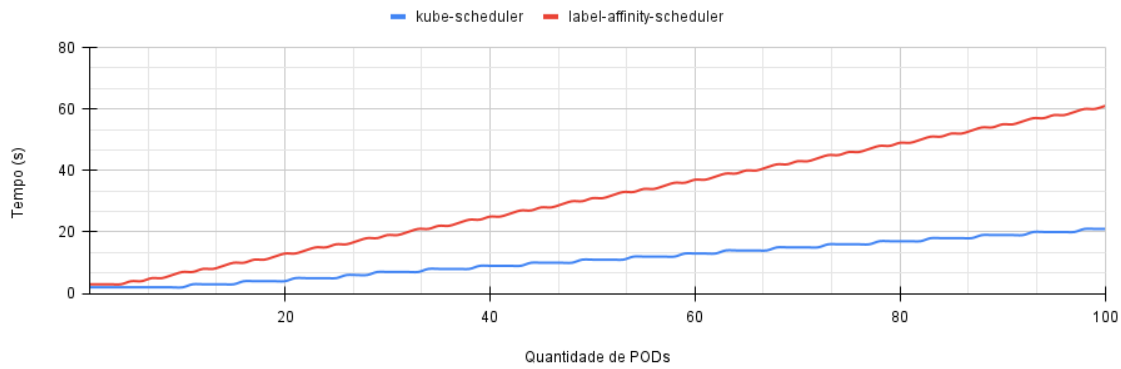

Figura 23 – Desempenho da distribuição dos *pods* da carga de trabalho "App H" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

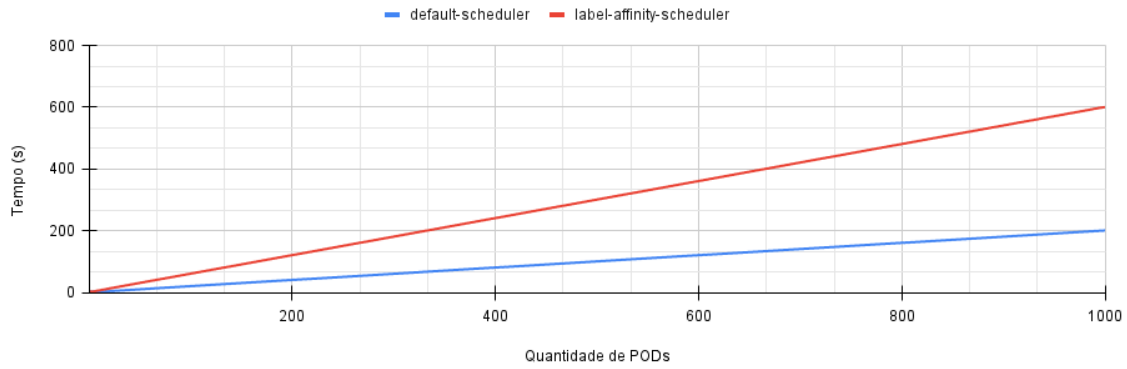

Figura 24 – Desempenho da distribuição dos *pods* da carga de trabalho "App I" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

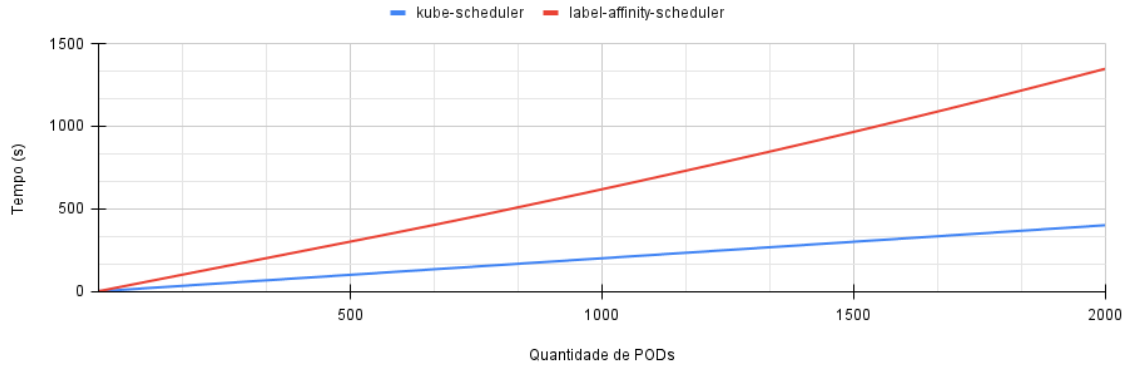

Figura 25 – Desempenho da distribuição dos *pods* da carga de trabalho "App J" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*).

Nestas visualizações de resultados é possível notar o comportamento interno da ferramenta *Kubernetes*, que envia os *pods* para definição à qual nó computacional será alocado em formato de "ondas", além disso, percebe-se a linearidade em ambos cenários.

A Tabela 18 apresenta o tempo médio (em segundos) decorrido pela ferramenta de orquestração para disponibilizar todas as instâncias da carga de trabalho e a mediana de cada um dos *pods*. A variação calculada referente ao tempo total do processo foi de 300,00%, 290,48%, 299,00% e 337,00% respectivamente para cada cenário, favorecendo o escalonador padrão *kube-scheduler* em relação ao escalonador personalizado *label-affinity-scheduler*.

Tabela 18 – Desempenho do processo de escalonamento das cargas de trabalho utilizadas no experimento 5.

Carga de Trabalho	***pods***	***kube-scheduler***		***label-affinity-scheduler***	
		Total (s)	**Média (s)**	**Total (s)**	**Média (s)**
App G	10	2	0.200	6	0.600
App H	100	21	0.210	61	0.610
App I	1000	201	0.201	601	0.601
App J	2000	400	0.200	1348	0.674

Esta diferença de desempenho para concluir a alocação de todos os *pods*, deve-se ao fato do escalonador personalizado realizar verificações em relação a afinidade dos rótulos definidos na carga de trabalho e nos nós computacionais, além disso, também é realizado uma análise de recursos computacionais ociosos, atribuindo um critério de desempate em cenários onde dois ou mais nós computacionais possuem o mesmo nível de afinidade.

A Figura 26 apresenta a variação de tempo em cada um dos cenários utilizados e sua definição de *pods*, permitindo uma análise do impacto ao utilizar o escalonador personalizado *label-affinity-scheduler* em diferentes volumes de instâncias.

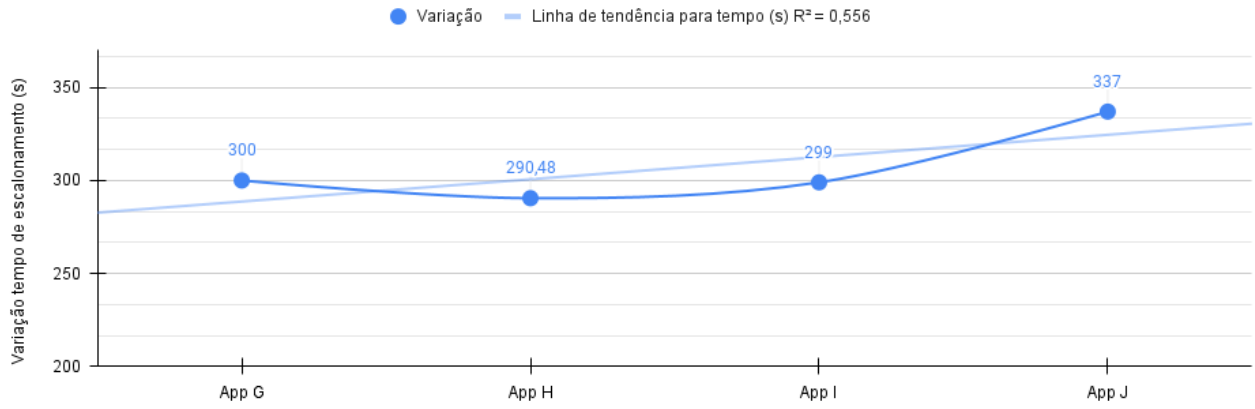

Figura 26 – Análise da variação de tempo e linha de tendência utilizando o escalonador personalizado *label-affinity-scheduler*.

O principal fato que corrobora com a depreciação de desempenho conforme aumenta-se a quantidade de *pods* está relacionado a análise de recursos computacionais livres em cada nó computacional, e conforme a quantidade de *pods* aumenta em cada um deles, maior é o esforço necessário na linha 5 do Algoritmo 2 em definir uma pontuação para o mesmo, logo, quanto maior a quantidade de nós computacionais e maior a quantidade de *pods* em execução, maior será o tempo necessário para realizar a escolha do nó para direcionamento do novo *pod* recebido para escalonamento.

Realizando o isolamento da carga de trabalho "App H", e submetendo-a à um ambiente *multi-cloud* composto por apenas 5 nós computacionais (ao invés dos 25 considerados originalmente) é possível atestar uma queda significativa no tempo necessário para escalonamento de todos os *pods*, de 61 segundos para 30 segundos, indicando uma melhora de 49,18%, como pode ser visto na Figura 27.

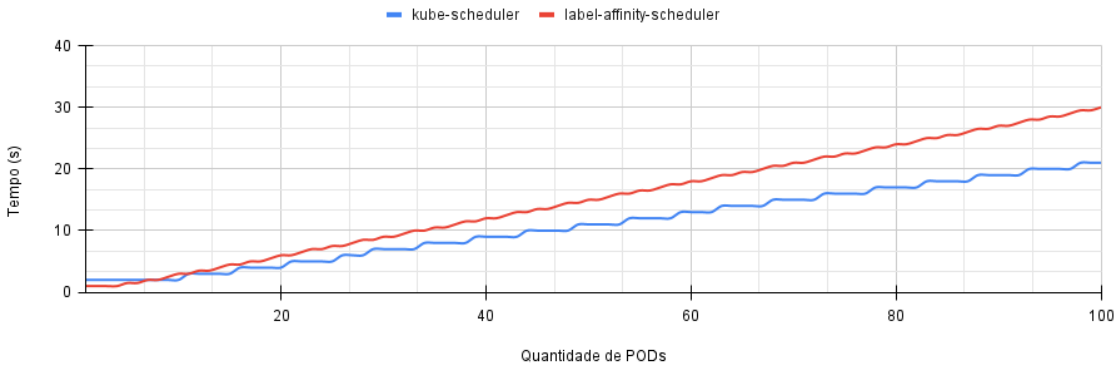

Figura 27 – Desempenho da distribuição dos *pods* da carga de trabalho "App H" utilizando escalonador padrão (*kube-scheduler*) e personalizado (*label-affinity-scheduler*) em um ambiente *multi-cloud* composto por 5 nós computacionais.

5.8 Discussão dos Resultados

A avaliação realizada sob o uso do escalonador *label-affinity-scheduler* em diferentes experimentos teve o intuito de demonstrar sua capacidade de adaptação a diferentes cenários compostos por restrições de negócio variadas. As cargas de trabalho que foram utilizadas também foram selecionadas para uma melhor coleta de dados de acordo com os objetivos aos quais estavam em avaliação, seja a qualidade da distribuição dos *pods* através do ecossistema *multi-cloud*, análise do comportamento durante cargas de trabalho de uso computacional intenso ou comparativo com propostas apresentadas no estado-da-arte.

Durante esta avaliação foi possível atestar os benefícios da adoção de um escalonador personalizado para direcionamento de cada uma das instâncias da aplicação, utilizando para isso uma linguagem de marcação, garantida através da funcionalidade de rótulos da ferramenta de orquestração *Kubernetes*. Unificar a administração de todos os nós computacionais de um ecossistema *multi-cloud* e oferecer alocação dinâmica de acordo com requisitos de usuários e de cada aplicação garantem uma prestação de serviço aprimorada.

Como ponto desfavorável, destaca-se o desempenho do escalonador *label-affinity-scheduler*, que conforme apresentado no Experimento 5, possui um maior custo de execução. Isso deve-se às etapas extras implementadas no *label-affinity-scheduler*, com ênfase na análise de afinidade dos rótulos entre as cargas de trabalho e os nós computacionais, e também pela verificação pertinente da quantidade de recurso computacional livre de cada um dos nós, de tal forma que a alocação fosse realizada aquele com maior capacidade livre.

5.9 Considerações Finais

Os experimentos apresentados tiveram o objetivo de demonstrar a utilização de um escalonador personalizado para ferramentas de orquestração de contêineres, alternativo

ao oferecido de forma padrão. Para isso, foi construído um ecossistema composto por nós computacionais distribuídos através de infraestruturas de diferentes fornecedores, caracterizando assim um ambiente *multi-cloud*. Sob tal ecossistema, foram utilizadas cargas de trabalho responsáveis por simular uso intensivo de recursos computacionais, como CPU e memória, buscando uma maior similaridade à cenários reais. Nestes experimentos foi possível atestar os benefícios e diferenças no uso do *label-affinity-scheduler*, gerenciando restrições pré-definidas nas cargas de trabalho durante o processo de escalonamento das instâncias computacionais destas aplicações.

A melhoria de qualidade obtida na alocação das instâncias computacionais e taxa de sucesso está ligada diretamente com a definição dos rótulos realizadas previamente em cada um dos nós computacionais, logo, também atrelada à contratação destes nós computacionais em harmonia com as cargas de trabalho a serem executadas, a fim de obter uma maior eficiência sob um menor investimento na contratação deste tipo de recurso.

Outras propostas sugeridas no estado-da-arte Capítulo 3 não foram abordadas nos experimentos, porém, existe a possibilidade de serem abrangidas pelo escalonador *label-affinity-scheduler*, como é o caso da proposta de (TORDSSON et al., 2012), em que sugerem a construção de um gerenciador de máquinas virtuais em um ambiente *multi-cloud*, para realizar a contratação destes recursos considerando requisitos de *hardware* (CPU, memória e região), a partir do menor preço praticado. Neste trabalho foi utilizada uma tabela de preços de um único fornecedor (*Amazon*), e conforme novas solicitações sejam recebidas, uma nova análise acontece para identificar o melhor local de contratação. Já no trabalho de (BAUR et al., 2018), é proposto o uso de uma linguagem declarativa para realizar a orquestração de cargas de trabalho, e quando necessário, realizar a contratação de máquinas virtuais para compor o ambiente *multi-cloud* e atender a nova requisição recebida.

Em ambos cenários envolve a definição de requisitos para as cargas de trabalho e o processo de orquestração das instâncias para nós computacionais que satisfaçam alguns critérios. Estender as funcionalidades do escalonador *label-affinity-scheduler* com a obtenção de dados máquinas virtuais dos fornecedores de forma *online*, utilizando *APIs* públicas disponíveis (*AWS*[15], *Azure*[16], *GCP*[17] e *Oracle*[18]), e configuração das credenciais das contas destas plataformas. Este nível de integração possibilitaria que, durante o processo de escalonamento, fossem realizadas análises dos nós computacionais disponíveis, e se necessário, realizar a contratação da máquina virtual, vinculação ao *cluster* gerenciado pelo orquestrador *Kubernetes* e posterior direcionamento do *pod* à ela.

[15] <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/price-changes.html>
[16] <https://docs.microsoft.com/en-us/rest/api/cost-management/retail-prices/azure-retail-prices>
[17] <https://cloud.google.com/billing/v1/how-tos/catalog-api>
[18] <https://apexapps.oracle.com/pls/apex/cetools/api/v1/products>

6 Conclusão

Nos ecossistemas de computação em nuvem, a adoção de contêineres tem se difundido de modo acelerado, principalmente pelas vantagens oferecidas em relação a implantação ágil de aplicações, portabilidade, isolamento e a melhor utilização dos recursos computacionais. Como visto, um problema particularmente importante a ser investigado é o escalonamento ou alocação de contêineres nos recursos disponíveis. Idealmente, conforme as aplicações conteinerizadas são enviadas para implantação, um sistema de orquestração deve alocar os respectivos contêineres de modo rápido em um dos recursos disponíveis, considerando fatores como a capacidade das máquinas disponíveis, custo de implantação, requisitos de desempenho e qualidade de serviço (*QoS*) da aplicação, tolerância a falhas e consumo de energia.

No entanto, as políticas de alocação presentes nos orquestradores atuais são completamente agnósticas no que diz respeito a demandas específicas das aplicações ou manutenção da *QoS*. Geralmente utilizam políticas que realizam a alocação das aplicações simplesmente espalhando os contêineres entre os nós de trabalho usando algoritmos como *Round-Robin* ou *First-Fit*. Alguns trabalhos na literatura propõem estratégias de escalonamento que levam em consideração algum aspecto em particular, mas são efetivas apenas nos casos onde um desses objetivos é considerado. No entanto, o plano de alocação de contêineres não é tão simples se múltiplos objetivos tiverem que ser considerados simultaneamente. O problema ainda se amplifica em um cenário multi-usuário, no qual cada um deles possui um conjunto de aplicações com requisitos de negócio distintos.

Sob este cenário, a contribuição deste trabalho foi apresentar uma solução que permite que o escalonamento de contêineres possa ser feito considerando um conjunto de requisitos de negócio personalizados de uma determinada aplicação ou de seu proprietário. Isso é feito, estendendo-se o esquema de rotulagem, já presente em alguns orquestradores, para permitir a atribuição de características aos nós computacionais e requisitos às aplicações, bem como realizar a vinculação das cargas de trabalho aos nós de maior afinidade. A proposta foi validada com a implementação de um escalonador personalizado chamado de *label-affinity-scheduler* para o orquestrador *Kubernetes*.

Por utilizar uma metodologia de marcação via rótulos, esta abordagem permite a organização e classificação dos nós computacionais e cargas de trabalho, além de oferecer uma utilização de forma simples e de fácil entendimento, inclusive garantindo que profissionais sem conhecimento específico da ferramenta de orquestração possam realizar a configuração de um escalonamento baseado nos requisitos de cada cliente e aplicação.

Em diferentes cenários onde o escalonador *label-affinity-scheduler* teve cargas de trabalho submetidas foi possível atestar a sua eficácia, garantindo de forma qualitativa que os requisitos de negócio sugeridos nas aplicações alvo dos testes, não fossem violados. Para uma maior confiabilidade de todos os cenários, foi constituído um ecossistema *multi-cloud*, composto por 25 nós computacionais distribuídos através de 4 fornecedores deste tipo de serviço, com diferentes configurações de *hardware* e localização geográfica, muito semelhante àquele encontrado em empresas que exploram este tipo de serviço. De qualquer forma, não há nenhum impedimento na utilização de outros fornecedores, bastando apenas que o nó computacional faça parte do ecossistema *multi-cloud* e tenha seus rótulos definidos.

A utilização do *label-affinity-scheduler* possibilitou um melhor controle do nó computacional alvo da alocação dos *pods* utilizando os rótulos, permitindo que diferentes objetivos fossem alcançados de forma independente entre cada um dos cenários, inclusive, todas as cargas de trabalho foram executadas no mesmo ambiente orquestrado, não havendo nenhum conflito entre as mesmas, apenas uma concorrência natural dos recursos computacionais.

Além dos cenários de teste com foco na avaliação qualitativa do processo de escalonamento, foi realizado um teste de desempenho, composto por quatro cenários distintos, utilizando diferentes tamanhos de instâncias para uma carga de trabalho, sendo eles, 10, 100, 1000 e 2000 instâncias (*pods*) necessários para uma mesma aplicação. Nestes cenários a variação calculada referente ao tempo total do processo foi de 300,00%, 290,48%, 299,00% e 337,00% respectivamente para cada um deles, favorecendo o escalonador padrão *kube-scheduler* em relação ao escalonador personalizado *label-affinity-scheduler*. Este impacto no desempenho deve-se às características implementadas, responsáveis por calcular a afinidade entre carga de trabalho e nó computacional, e análise da capacidade ociosa de cada um dos nós, com o objetivo de privilegiar aqueles com a maior capacidade de *hardware* livre, abrindo margem para pesquisa de abordagens alternativas à tais processos.

6.1 Trabalhos Futuros

O direcionamento futuro para melhorias no escalonador personalizado *label-affinity-scheduler* envolve aspectos relacionados ao seu desempenho durante o processo de definição do nó computacional alvo a ser vinculado os *pods* recebidos para alocação, a automatização do preenchimento de alguns rótulos utilizando para isso *APIs* disponíveis para tal finalidade e o provisionamento dinâmico de nós computacionais que compõem o ecossistema *multi-cloud*, tornando o ambiente flexível e apto a escalonar cargas de trabalho com quaisquer características que envolvam configuração de *hardware* e localização geográfica.

Para obter a melhoria de desempenho, deve-se isolar os eventos responsáveis por coletar as estatísticas de *hardware* de cada nó computacional que compõe o ecossistema

multi-cloud. Em análises de comportamento durante a execução dos cenários de teste, percebeu-se uma grande complexidade durante a realização desta tarefa, sendo depreciada conforme aumenta-se a quantidade de *pods* vinculados a cada nó computacional. Como alternativas para contornar este comportamento e oferecer um incremento de desempenho ao escalonador está a possibilidade de construção de um *cache* para armazenar dados computacionais de cada nó do *cluster*, ou uma mudança de paradigma, onde o escalonador padrão *kube-scheduler* fará a disponibilização inicial, e em seguida o escalonador personalizado irá disparar um evento de re-escalonamento, direcionando a instância da aplicação para um nó computacional mais indicado. Neste novo paradigma, o impacto para o usuário final será nulo, pois o orquestrador *Kubernetes* trabalha com um conceito para atualizações chamado *rolling update*[1], onde os novos *pods* são disponibilizados, para aí então os obsoletos serem removidos, e nos cenários em que há algum tipo de acesso externo, o serviço de *DNS* embarcado garante o direcionamento das requisições entre estes *pods*, evitando a indisponibilidade de acesso.

Outra dificuldade existente durante a operação em um ecossistema *multi-cloud* está relacionada a diferentes políticas de preço praticadas pelos diversos fornecedores. Regularmente, tabelas de preço passam por revisão, e como estes preços são tratados via rótulos vinculados aos nós computacionais, acabam não refletindo a realidade e se tornando obsoletos. Uma possível melhoria está relacionada a utilização de *APIs* para coleta e atualização de tais informações diretamente dos fornecedores, como é o caso da *AWS*[2], *Azure*[3], *GCP*[4] e *Oracle*[5].

Ainda no contexto das *APIs* públicas para acesso aos produtos ofertados pelos fornecedores de nuvem computacional, está a possibilidade de realizar o provisionamento dinâmico de nós computacionais de acordo com as necessidades explícitas definidas nos rótulos das cargas de trabalho. Atualmente, o condicionante de sucesso durante o escalonamento utilizando o *label-affinity-scheduler* está diretamente relacionado à existência de nós que satisfaçam tais critérios, caso contrário, não será possível disponibilizar os *pods* da aplicação, ou ainda, se os nós que satisfizerem os critérios não possuírem recursos de *hardware* para executar todos os *pods* requisitados, o escalonamento também não será bem sucedido. Ambos cenários podem ser contornados através do consumo de barramentos das *APIs* públicas para realizar o provisionamento de nós computacionais quando houver necessidade.

[1] <https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro>
[2] <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/price-changes.html>
[3] <https://docs.microsoft.com/en-us/rest/api/cost-management/retail-prices/azure-retail-prices>
[4] <https://cloud.google.com/billing/v1/how-tos/catalog-api>
[5] <https://apexapps.oracle.com/pls/apex/cetools/api/v1/products>

Referências

ANTONESCU, A.; ROBINSON, P.; BRAUN, T. Dynamic topology orchestration for distributed cloud-based applications. Proceedings - IEEE 2nd Symposium on Network Cloud Computing and Applications, NCCA 2012, 12 2012. Citado na página 36.

ARUNDEL, J.; DOMINGUS, J. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud. O'Reilly Media, Incorporated, 2019. ISBN 9781492040767. Disponível em: <https://books.google.com.br/books?id=xZXfuQEACAAJ>. Citado na página 33.

BAIER, J.; SAYFAN, G.; WHITE, J. The Complete Kubernetes Guide: Become an expert in container management with the power of Kubernetes. Packt Publishing, 2019. ISBN 9781838647704. Disponível em: <https://books.google.com.br/books?id=0JmZDwAAQBAJ>. Citado na página 34.

BALALA, M. How a Multi-cloud Strategy Enables Digital Transformation. O'Reilly Media, 2018. Disponível em: <https://books.google.com.br/books?id=XlqKswEACAAJ>. Citado na página 41.

BAUR, D. et al. A provider-agnostic approach to multi-cloud orchestration using a constraint language. In: Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Press, 2018. (CCGrid '18), p. 173–182. ISBN 9781538658154. Disponível em: <https://doi.org/10.1109/CCGRID.2018.00032>. Citado 5 vezes nas páginas 16, 43, 45, 48 e 91.

BOHLI, J.-M. et al. Security and privacy-enhancing multicloud architectures. Dependable and Secure Computing, IEEE Transactions on, v. 10, p. 212–224, 07 2013. Citado na página 24.

CARVALHO, L.; ARAUJO, A. Performance comparison of terraform and cloudify as multicloud orchestrators. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). [S.l.: s.n.], 2020. p. 380–389. Citado na página 42.

CASALICCHIO, E. Autonomic orchestration of containers: Problem definition and research challenges. 10th EAI International Conference on Performance Evaluation Methodologies and Tools, ACM, 5 2017. Citado 4 vezes nas páginas 8, 36, 39 e 44.

CASALICCHIO, E. Container orchestration: A survey. In: _____. Systems Modeling: Methodologies and Tools. [S.l.]: Springer, 2019. p. 221–235. ISBN 978-3-319-92377-2. Citado 3 vezes nas páginas 37, 38 e 45.

CASQUERO, O. et al. Distributed scheduling in kubernetes based on mas for fog-in-the-loop applications. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). [S.l.: s.n.], 2019. p. 1213–1217. Citado 2 vezes nas páginas 8 e 55.

CHANDRASEKARAN, K. Essentials of Cloud Computing. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2014. ISBN 1482205432. Citado 3 vezes nas páginas 19, 20 e 21.

DHAR, R. Comparative evaluation of Virtual Environments: Virtual Machines and Containers. Tese (Theses) — University of Dublin, Trinity College, 08 2016. Citado na página 28.

EDER, M. Hypervisor- vs. container-based virtualization. Seminar Future Internet, Chair for Network Architectures and Services, Department of Computer Science, Technische Universität München, 2016. Disponível em: <http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2016-07-1/NET-2016-07-1_01.pdf>. Citado na página 31.

GROZEV, N.; BUYYA, R. Regulations and latency-aware load distribution of web applications in multi-clouds. J. Supercomput., Kluwer Academic Publishers, Hingham, MA, USA, v. 72, n. 8, p. 3261–3280, ago. 2016. ISSN 0920-8542. Disponível em: <http://dx.doi.org/10.1007/s11227-016-1735-6>. Citado na página 24.

GUERRERO, C.; LERA, I.; JUIZ, C. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. The Journal of Supercomputing, v. 74, 07 2018. Citado 2 vezes nas páginas 43 e 45.

HOHPE, G. Cloud Strategy An Architecture Guide to Successful Cloud Migration. Leanpub, 2020. Disponível em: <https://leanpub.com/cloudstrategy>. Citado 5 vezes nas páginas 8, 10, 24, 25 e 27.

HURWITZ, J. et al. Cloud Computing For Dummies. [S.l.]: For Dummies, 2009. ISBN 0470484705. Citado na página 21.

HWANG, K.; DONGARRA, J.; FOX, G. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Elsevier Science, 2013. ISBN 9780128002049. Disponível em: <https://books.google.com.br/books?id=IjgVAgAAQBAJ>. Citado na página 29.

JAMES, A.; SCHIEN, D. A low carbon kubernetes scheduler. In: ICT4S. [S.l.: s.n.], 2019. Citado 9 vezes nas páginas 16, 40, 45, 47, 71, 72, 81, 82 e 84.

KLAFFENBACH, F.; KLEIN, M.; SUNDARESAN, S. Multi-Cloud for Architects: Grow your IT business by means of a multi-cloud strategy. Packt Publishing, 2019. ISBN 9781788627801. Disponível em: <https://books.google.com.br/books?id=DGSGDwAAQBAJ>. Citado 2 vezes nas páginas 22 e 33.

KUBERNETES. Horizontal Pod Autoscaler. 2019. Acessado em: 22 fev. 2021. Disponível em: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale>. Acesso em: 22 fev. 2021. Citado na página 52.

KUBERNETES. Ports and Protocols. 2019. Acessado em: 12 mar. 2022. Disponível em: <https://kubernetes.io/docs/reference/ports-and-protocols>. Acesso em: 12 mar. 2022. Citado 4 vezes nas páginas 10, 11, 101 e 102.

KUBERNETES. Assigning Pods to Nodes. 2020. Acessado em: 14 mar. 2021. Disponível em: <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node>. Acesso em: 14 mar. 2021. Citado 4 vezes nas páginas 8, 52, 53 e 54.

KUBERNETES. Kubernetes Scheduler. 2020. Disponível em: <https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler>. Citado 2 vezes nas páginas 16 e 47.

KUBERNETES. Labels and Selectors. 2020. Acessado em: 17 jan. 2021. Disponível em: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels>. Acesso em: 17 jan. 2021. Citado 2 vezes nas páginas 57 e 62.

KUBERNETES. Understanding Kubernetes Objects. 2020. Acessado em: 20 ago. 2021. Disponível em: <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects>. Acesso em: 20 ago. 2021. Citado na página 62.

LEAVITT, N. Is cloud computing really ready for prime time? IEEE Computer, v. 42, p. 15–20, 01 2009. Citado na página 21.

MALATHI, M. Cloud computing concepts. 04 2011. Citado na página 19.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing. Gaithersburg, MD, 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>. Citado na página 18.

MENOUER, T. Kcss: Kubernetes container scheduling strategy. The Journal of Supercomputing, 09 2020. Citado 2 vezes nas páginas 15 e 16.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal, v. 2014, 03 2014. Citado na página 30.

MORENO-VOZMEDIANO, R. et al. Orchestrating the deployment of high availability services on multi-zone and multi-cloud scenarios. Journal of Grid Computing, v. 16, 03 2018. Citado 2 vezes nas páginas 43 e 45.

MOUAT, A. Using Docker. [S.l.]: O'Reilly, 2015. ISBN 9781491915769. Citado 4 vezes nas páginas 30, 33, 34 e 51.

NANDA, S.; CHIUEH, t.-c. A Survey on Virtualization Technologies. [S.l.], 2005. Citado na página 28.

NGUYEN, T. L. Fast delivery of virtual machines and containers : understanding and optimizing the boot operation Tese (Theses) — Ecole nationale supérieure Mines-Télécom Atlantique, 09 2019. Disponível em: <https://tel.archives-ouvertes.fr/tel-02418752>. Citado na página 27.

NGUYEN, T.-T. et al. Horizontal pod autoscaling in kubernetes for elastic container orchestration. Sensors, v. 20, p. 4621, 08 2020. Citado na página 15.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective. J. Cloud Comput., Hindawi Publishing Corp., New York, NY, United States, v. 5, n. 1, p. 54:1–54:18, dez. 2016. ISSN 2192-113X. Disponível em: <https://doi.org/10.1186/s13677-016-0054-z>. Citado na página 22.

PALAN, V. H. A Novel Approach to Containerize Existing Applications. Tese (Theses) — Florida Institute of Technology, 06 2018. Disponível em: <http://hdl.handle.net/11141/2518>. Citado na página 33.

PRASAD, D. Comparison Type 1 vs Type 2 Hypervisor. 2014. Acessado em: 12 abr. 2020. Disponível em: <https://www.golinuxhub.com/2014/07/comparison-type-1-vs-type-2-hypervisor.html>. Acesso em: 12 abr. 2020. Citado 3 vezes nas páginas 8, 28 e 29.

RAJ, P.; RAMAN, A. Automated multi-cloud operations and container orchestration. In: _____. Software-Defined Cloud Centers: Operational and Management Technologies and Tools. Cham: Springer International Publishing, 2018. p. 185–218. ISBN 978-3-319-78637-7. Disponível em: <https://doi.org/10.1007/978-3-319-78637-7_9>. Citado 5 vezes nas páginas 8, 23, 35, 37 e 42.

ROCHA, I. et al. Heats: Heterogeneity-and energy-aware task-based scheduling. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). [S.l.: s.n.], 2019. p. 400–405. Citado 3 vezes nas páginas 16, 40 e 45.

RODRIGUEZ, M. A.; BUYYA, R. Containers orchestration with cost-efficient autoscaling in cloud computing environments. ArXiv, abs/1812.00300, 2018. Citado 9 vezes nas páginas 15, 40, 47, 71, 72, 77, 78, 79 e 80.

ROSSI, F. et al. Geo-distributed efficient deployment of containers with kubernetes. Computer Communications, v. 159, p. 161 – 174, 2020. ISSN 0140-3664. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0140366419317931>. Citado 4 vezes nas páginas 16, 40, 45 e 47.

S, D.; CHOLLI, N. Green cloud computing: Redefining the future of cloud computing. International Research Journal on Advanced Science Hub, v. 3, p. 12–19, 07 2021. Citado na página 82.

SAYFAN, G. Mastering Kubernetes: Master the Art of Container Management by Using the Power of Kubernetes, 2nd Edition. Packt Publishing, 2018. ISBN 9781788999786. Disponível em: <https://books.google.com.br/books?id=V7lztgEACAAJ>. Citado 3 vezes nas páginas 34, 35 e 101.

SCHOLL, B.; SWANSON, T.; JAUSOVEC, P. Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications. O'Reilly Media, Incorporated, 2019. (System administration). ISBN 9781492053828. Disponível em: <https://books.google.com.br/books?id=DDpwwgEACAAJ>. Citado 4 vezes nas páginas 8, 31, 32 e 37.

SOLTESZ, S. et al. Container-basedoperatingsystemvirtualization: Ascalable,high-performancealternativetohypervisors. In: Container-basedOperatingSystemVirtualization: AScalable,High-performanceAlternativetoHypervisors. [S.l.: s.n.], 2007. v. 41, p. 275–287. Citado na página 31.

TANENBAUM, A. S.; BOS, H. Modern Operating Systems. 4th. ed. USA: Prentice Hall Press, 2014. ISBN 013359162X. Citado 2 vezes nas páginas 29 e 30.

TOMARCHIO, O.; CALCATERRA, D.; MODICA, G. Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. Journal of Cloud Computing, v. 9, p. 1–24, 2020. Citado 2 vezes nas páginas 42 e 47.

TORDSSON, J. et al. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems, v. 28, n. 2, p. 358 – 367, 2012. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X11001373>. Citado 3 vezes nas páginas 43, 45 e 91.

TOSATTO, A.; RUIU, P.; ATTANASIO, A. Container-based orchestration in cloud: State of the art and challenges. 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, p. 70–75, 07 2015. Citado na página 36.

TRUYEN, E. et al. A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks. 2020. Citado na página 38.

VARGHESE, B.; BUYYA, R. Next generation cloud computing: New trends and research directions. Future Generation Computing Systems, Elsevier, v. 79‘, n. 3, p. 849–861, 9 2017. ISSN 0167-739X. Citado na página 23.

WANG, Z. et al. Research and implementation of scheduling strategy in kubernetes for computer science laboratory in universities. Information, v. 12, p. 16, 01 2021. Citado na página 55.

WEERASIRI, D. et al. A taxonomy and survey of cloud resource orchestration techniques. ACM Comput. Surv., Association for Computing Machinery, New York, NY, USA, v. 50, n. 2, maio 2017. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3054177>. Citado na página 36.

YADAV, A.; GARG, M.; MEHRA, R. Docker containers versus virtual machine-based virtualization: Proceedings of iemis 2018, volume 3. In: _____. [S.l.]: Springer US, 2019. p. 141–150. ISBN 978-981-13-1500-8. Citado 3 vezes nas páginas 8, 30 e 32.

ZHONG, Z.; BUYYA, R. A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. ACM Transactions on Internet Technology, v. 20, p. 1–24, 04 2020. Citado na página 15.

Apêndices

APÊNDICE A – Configuração de Rede no Ambiente *Multi-Cloud*

Uma das tarefas necessárias a serem executadas após a configuração inicial do ambiente orquestrado utilizando a ferramenta *Kubernetes* envolve a definição e uso de um *plugin* para realizar o gerenciamento de rede e comunicação entre os contêineres.

Também conhecidos como CNI (*Container Network Interface*), estes *plugins* garantem a comunicação e acessibilidade dos contêineres entre si, inclusive, alguns dos *plugins* disponíveis no mercado oferecem recursos para configuração de regras de segurança de rede, elevando as possibilidade o gerenciamento e administração destas configurações, (SAYFAN, 2018). Dentre os mais populares, destacam-se o *Weave*[1], *Flannel*[2] e o *Project Calico*[3].

Ao construir um ambiente orquestrado composto por quatro diferentes fornecedores (*Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)*, *Microsoft Azure* e *Oracle Cloud Infrastructure (OCI)*) alguns desafios envolveram a configuração dos parâmetros de rede, para que a comunicação entre os nós computacionais do *cluster* ocorresse conforme o esperado.

Cada um dos fornecedores possui suas características e definições próprias em relação às configurações de *subnet*, regras de entrada (*ingress*) e saída (*egress*) do tráfego de dados, logo, há a necessidade de parametrizá-las para que a ferramenta de orquestração, através do *plugin* de rede definido, estabelecer a comunicação entre os nós computacionais, bem como distribuir os contêineres através deles.

As Tabelas 19 e 20, adaptadas de (KUBERNETES, 2019b), relacionam a lista de portas utilizadas na comunicação entre os nós computacionais que constituem o *cluster* orquestrado, tais portas devem ser configuradas individualmente dentro das contas administrativas de cada fornecedor de nuvem computacional.

Tabela 19 – Portas de comunicação utilizadas para tráfego de informações entre os contêineres. Adaptada de (KUBERNETES, 2019b)

Protocolo	**Direção**	**Porta(s)**	**Propósito**
TCP	Entrada	10250	*Kubelet API*
TCP	Entrada	30000-32767	*NodePort Services*

[1] <https://www.weave.works>
[2] <https://github.com/flannel-io/flannel>
[3] <https://www.tigera.io/project-calico>

Tabela 20 – Portas de comunicação utilizadas para tráfego de informações entre os nós computacionais com a instância principal do *Kubernetes*. Adaptada de (KUBERNETES, 2019b)

Protocolo	**Direção**	**Porta(s)**	**Propósito**
TCP	Entrada	6443	*Kubernetes API*
TCP	Entrada	2379-2380	*etcd server client API*
TCP	Entrada	10250	*Kubelet API*
TCP	Entrada	10257	*kube-scheduler*
TCP	Entrada	10259	*kube-controller-manager*

A realização desta configuração no ecossistema dos fornecedores está relacionada à parâmetros de rede. A Tabela 21 apresenta a localização destes acessos dentro dos ambientes dos fornecedores utilizados.

Tabela 21 – Portas de comunicação utilizadas para tráfego de informações entre os contêineres. Adaptada de (KUBERNETES, 2019b)

Fornecedor	**Recurso**
Amazon Web Services (AWS)	*Virtual Private Cloud > Network ACL > Inbound/Outbound Rules*
Google Cloud Platform (GCP)	*VPC Network > Firewall > Rules*
Microsoft Azure	*Network > Network Security Groups > Settings > Inbound/Outbound Security Rules*
Oracle Cloud Infrastructure (OCI)	*Networking > Virtual Cloud Networks > Subnet Details > Security List > Rules*