Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Implantação do Knative em um cluster de Kubernetes utilizando a Google Cloud

Giovanni Sávio Pereira

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:

Marlon Paolo Lima - Orientador

COORIENTAÇÃO:

Alberto Oliveira da Silva - Coorientador

Dezembro, 2019
João Monlevade–MG

Giovanni Sávio Pereira

Implantação do Knative em um cluster de Kubernetes utilizando a Google Cloud

Orientador: Marlon Paolo Lima - Orientador

Coorientador: Alberto Oliveira da Silva - Coorientador

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Dezembro de 2019

P436i Pereira, Giovanni Sávio.
Implantação do Knative em um cluster de Kubernetes utilizando a Google Cloud [manuscrito] / Giovanni Sávio Pereira. - 2019.

52f.:

Orientador: Prof. Dr. Marlon Paolo Lima.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Software - Desenvolvimento. 2. Computação sem servidor. 3. Internet. I. Lima, Marlon Paolo. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 004.777

Catalogação: ficha.sisbin@ufop.edu.br

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Giovanni Savio Pereira

Implantação do Knative em um cluster de Kubernetes utilizando a Google Cloud

Membros da banca

Alberto Oliveira da Silva (Coorientador) - Graduação - Membro externo
Prof. Bruno Cerqueira Hott - Mestrado - DECSI/UFOP
Prof. Theo Silva Lins - Mestrado - DECSI/UFOP
Prof. Fabianni Roberto Teles - Mestrado - DECSI/UFOP

Versão final
Aprovado em 06 de Dezembro de 2019

De acordo

Prof. Dr. Marlon Paolo Lima

Documento assinado eletronicamente por **Marlon Paolo Lima**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 23/12/2019, às 17:35, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0030192** e o código CRC **0582DA17**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: - www.ufop.br

Este trabalho é dedicado aos meus pais, por terem me dado a oportunidade que eles nunca tiveram.

Agradecimentos

Agradeço aos meus pais, Geraldo e Vicentina, ao meu irmão Samuel pelo apoio durante todos esses anos, a Rafaela por nunca ter me deixado desistir, a todos os meus amigos por estarem sempre comigo nos bons e maus momentos e a República Vuco Vuco por toda diversão, aprendizados e irmandade que vou levar comigo por toda minha vida.

Gostaria de agradecer também aos meus orientadores Marlon e Alberto, aos bons professores que tive durante a graduação, por todos os ensinamentos transmitidos e a Stilingue por ter me dado a oportunidade de realizar esse estudo.

Resumo

Esta monografia apresenta um estudo da implantação do Knative utilizando a plataforma Google Cloud. Ao longo deste trabalho, discutiu-se sobre a criação de um *cluster* de Kubernetes utilizando a *cloud* do Google, incluindo particularidades sobre o processo de instalação e configuração. Além disso, algumas aplicações já existentes foram adaptadas para serem implantadas utilizando o Knative. Diferentes testes foram executados em duas aplicações reais, considerando três cenários distintos, com o intuito de avaliar a eficiência da abordagem proposta. Os resultados obtidos demonstram que a partir do uso do Knative, juntamente com a Google Cloud, é possível obter soluções que podem auxiliar na redução de custos operacionais, mantendo o desempenho esperado. Entretanto, uma análise mais aprofundada deve ser realizada considerando aplicações específicas que exijam grandes cargas de trabalho, visto que o Knative está completando um ano de criação no momento da redação desta monografia.

Palavras-chaves: Knative, Google Cloud, Kubernetes, Infraestrutura.

Abstract

This monograph presents a study of Knative deployment using the Google Cloud platform. Throughout this work, it is discussed the creation of a Kubernetes cluster using the cloud of Google, including particularities about the installation and configuration process. Besides, some existing applications have been adapted to be deployed using Knative. Different tests were performed in two real applications, considering three distinct scenarios, in order to evaluate the efficiency of the proposed approach. The results show that by using Knative associated with Google Cloud, it is possible to obtain solutions that can help reduce operating costs while maintaining expected performance. However, further analysis should be done to specific applications that require large workloads, as the Knative is completing one year of creation at the time of writing this monograph.

Key-words: Knative, Google Cloud, Kubernetes, Infrastructure.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CPU	Central Processing Unit
FaaS	Function as a Service
GB	Gigabyte
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
IaaS	Infrastructure as a Service
IP	Internet Protocol
KPA	Knative Pod Autoscaler
PaaS	Platform as a Service
RAM	Random access memory
SaaS	Software as a Service
TI	Tecnologia da Informação
vCPU	Virtual Central Processing Unit
VM	Maquina Virtual

Sumário

1 Introdução

A forma de se pensar em infraestrutura de Tecnologia da Informação (TI) mudou muito nos últimos anos devido à grande movimentação das estruturas de servidores locais para nuvens. Uma das definições mais utilizadas para computação em nuvem (*cloud computing*) foi determinada pelo *National Institute of Standard and Technology* (NIST) como um modelo que permite acesso via rede a um conjunto compartilhado de recursos computacionais (redes, servidores, armazenamento, aplicações e serviços), tendo, então, a característica de serem provisionados e disponibilizados, rapidamente, com o mínimo de esforço ou interação com o provedor de serviço (RIBAS et al., 2014).

Esse tipo de configuração permite uma manutenção mais fácil das máquinas e equipamentos, segurança contra desastres naturais e falhas de componentes físicos, além de ser facilmente escalável e flexível, o que é uma das principais vantagens e necessidades para infraestrutura encontradas hoje. Apesar de todos os benefícios descritos, o custo estrutural para se obtê-los é muito alto. Deste modo, é preciso pensar em alternativas viáveis que possam gerenciar melhor os recursos mantidos na nuvem com o objetivo de reduzir tais custos.

A infraestrutura local enfrenta várias dificuldades que não ocorrem ao utilizar a nuvem, como: alto investimento inicial com servidores físicos, energia elétrica, espaço físico, refrigeração, segurança com a manutenção e disponibilidade dos dados, fatos que não ocorrem quando se utiliza infraestrutura em nuvem, uma vez que todos estes pontos são responsabilidade do provedor. Ainda, em relação à infraestrutura local, existem dificuldades relacionadas com a escalabilidade, recursos, que seriam utilizados apenas em situação de pico, ficam sub-alocados durante momentos de baixa demanda, desperdiçando recursos computacionais.

A nuvem ainda trás mais vantagens além da alta escalabilidade, como abstração da complexidade de rede, ausência da necessidade de manutenção de equipamentos, menor número de profissionais responsáveis pelo trabalho e a garantia da integridade dos dados. O mercado atual é dinâmico, por conta disso, é de extrema importância que se tenha rapidez nas respostas às demandas de requisições; esta característica pode ser representada através da capacidade de escalabilidade dos recursos exigidos por uma aplicação.

Todas essas vantagens podem resultar em um elevado custo financeiro, dado que o gasto para a utilização da nuvem é baseado na quantidade de recursos alocados (reservados ou contratados). Por isso, é preciso encontrar alternativas para usar os recursos disponíveis na nuvem de forma mais inteligente e sem perder desempenho, evitando manter recursos ociosos em situações nas quais não existem tantas demandas e possibilitando, assim, a

redução dos altos custos com a nuvem.

Para ajudar a gerenciar esses recursos, existem os *containers*, uma tecnologia que permite empacotar e isolar aplicações com todas as configurações e arquivos necessários para sua execução. Esta tecnologia possibilita a fácil migração de aplicações em *containers* de um ambiente para outro, seja ele de desenvolvimento, teste ou produção (REDHAT, 2019a).

No entanto, o gerenciamento dos *containers* é muito trabalhoso e, com isso, surgiu a demanda de gerenciá-los de forma mais automática. Assim, surge o Kubernetes. Segundo RedHat (2019c), o Kubernetes, também conhecido como K8s, é um sistema de orquestração de *containers open source* que automatiza a implantação, o dimensionamento e a gestão de aplicações em *containers*.

O Kubernetes foi originalmente projetado pelo Google e agora é mantido pela *Cloud Native Computing Foundation*. Um *cluster* de Kubernetes é composto por *nodes*, que são máquinas (físicas ou virtuais) tendo como principal função executar os *pods*, que são a menor unidade de trabalho dentro do Kubernetes, podendo, cada um deles, ser composto por um ou mais *containers*. Uma vez que o Kubernetes já se apresenta como uma tecnologia consolidada e que funciona muito bem, surge agora a demanda por reduzir os custos com a nuvem. Como alternativa para essa demanda, tem-se o Knative.

O Knative foi anunciado na Google Cloud Next 2018 como "uma plataforma baseada em Kubernetes[1] para *build, deploy* e gerenciamento de *serverless workloads* modernos"(BRYANT, 2018). Desenvolvido pelo Google em parceria com a Pivotal, IBM, Red Hat e a SAP, possui entre suas principais funções, o *scale to zero*[2] de aplicações *cloud native*. O Knative foi construído utilizando o Kubernetes e o Istio[3], desenhado para contabilizar personas interagindo com o framework, incluindo desenvolvedores, operadores e provedores de plataforma (BRYANT, 2018).

Dessa forma, analisar o desempenho e a maturidade dessa tecnologia em um ambiente de produção é de grande importância para sabermos se esta ferramenta pode ser utilizada no dia a dia de pequenas, médias e grandes empresas. Com a implantação do Knative, através do componente *serving*, responsável por fazer o *scale to zero* das aplicações, espera-se um melhor aproveitamento dos recursos do *cluster* e, com isso, uma redução considerável dos custos com a nuvem.

Este trabalho foi realizado na Stilingue[4], uma empresa de Inteligência Artificial que foca em gestão de marcas em tempo real através da coleta, processamento e enriquecimento de dados disponíveis na Internet como postagens em redes sociais e publicações em portais

[1] Sistema *open source*, que automatiza e orquestra aplicações em *containers*
[2] Possibilita reduzir o número de instâncias de aplicações que não estão sendo utilizadas, até chegar a zero, ou seja, a aplicação será desligada.
[3] Plataforma aberta para conectar e proteger microserviços.
[4] Stilingue Inteligência Artificial, site: https://stilingue.com.br

de notícias. A Stilingue utiliza-se principalmente de Processamento de Linguagem Natural (NLP) desenvolvido na própria empresa para resumir o alto volume de informações disponíveis na Internet.

1.1 Objetivos

O objetivo geral desse trabalho é analisar o uso do Knative como uma ferramenta alternativa para auxiliar no problema do alto consumo de recursos e custos elevados de um *cluster* de Kubernetes implementado utilizando a nuvem do Google.

1.1.1 Objetivos Específicos

Para que o objetivo principal descrito acima seja alcançado, é fundamental que durante o processo, os seguintes objetivos específicos sejam cumpridos:

- Criar um cluster de Kubernetes utilizando a nuvem do Google;
- Instalar e configurar o Knative no *cluster* de Kubernetes criado;
- Adaptar aplicações existentes para serem implantadas utilizando o Knative;
- Analisar o impacto gerado pelas aplicações implantadas utilizando o Knative, no que se refere à alocação de recursos, desempenho e redução de custos no *cluster* de Kubernetes.

1.2 Justificativa

O presente trabalho tem como justificativa demonstrar os benefícios e os impactos da implantação do Knative para otimizar a utilização de recursos de um *cluster* de Kubernetes em um ambiente de computação em nuvem.

1.3 Estrutura do Trabalho

O conteúdo deste trabalho foi organizado em cinco capítulos: Introdução, Referencial Teórico, Metodologia, Resultados e Considerações finais. O primeiro capítulo trata da introdução ao tema, os objetivos e a justificativa. No segundo capítulo, é apresentada uma revisão da literatura sobre os conceitos relacionados a computação em nuvem, tecnologia de *containers*, Kubernetes, *serverless* e o estado da arte sobre Knative. O terceiro capítulo relata o processo e o ambiente de desenvolvimento, detalhes da implantação da tecnologia e adaptação das aplicações utilizadas. O quarto capítulo apresenta os resultados obtidos

com o trabalho e por fim, no quinto capítulo são apresentadas as considerações finais e as melhorias subsequentes para os trabalhos futuros.

2 Referencial Teórico

Este capítulo contempla a fundamentação teórica para a implementação do trabalho, abordando os conceitos, ferramentas e tecnologias utilizadas.

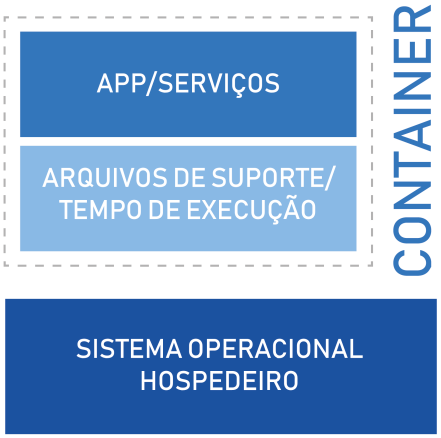
2.1 Tecnologia de *Containers*

A tecnologia de *containers* permite empacotar e isolar aplicações com todas as configurações e arquivos necessários para sua execução, o que possibilita a fácil migração de aplicações em *containers* de um ambiente para outro, seja ele de desenvolvimento, teste ou produção (REDHAT, 2019a). Para instanciar um *container* é necessário ter uma imagem que é um pacote de arquivos que inclui o executável de uma aplicação e todas as suas dependências.

As aplicações estão cada dia mais complexas e a agilidade no desenvolvimento de software é cada vez mais exigida. Com isso, a pressão sobre as equipes de desenvolvimento, infraestrutura e processos aumenta consideravelmente. Assim, os *containers* reduzem a possibilidade de problemas e possibilitam iterações mais rápidas em ambientes diferentes.

A estrutura de um *container* é composta de aplicativos, serviços e dos arquivos necessários para sua execução no sistema operacional hospedeiro, como é possível observar na Figura 1.

Figura 1 – Estrutura de um container



Fonte: Imagem adaptada de https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container

Ao fazer uso de *containers*, o desenvolvedor pode se manter focado no mais importante: o processo de desenvolvimento, propriamente dito, sem precisar se preocupar com a

infraestrutura.

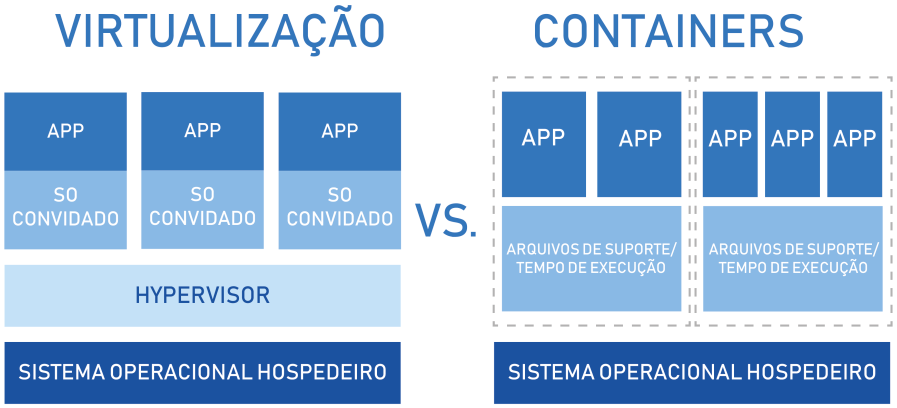
A tecnologia de *containers* é *open source*, uma das suas principais vantagens está diretamente relacionada à segurança da informação, pois permite proteger a infraestrutura, garantindo a sua confiança e escalabilidade.

2.1.1 Diferença entre Virtualização e *Containers*

A virtualização possibilita a execução de forma simultânea de sistemas operacionais diferentes por meio de máquinas virtuais, todas utilizando o mesmo tipo de *hardware*. Isso é possível graças à camada de *hypervisor* que emula e permite a configuração do *hardware*, como pode ser visto na Figura 2. Porém, essa é uma alternativa que exige uma quantidade considerável de recursos da máquina hospedeira. Já os *containers* trabalham de forma diferente, não possuem a camada de *hypervisor*, compartilhando, então, o mesmo *kernel* [1] do sistema operacional. Os processos da aplicação ficam isolados do restante do sistema, o que permite que a execução possa ser feita de forma paralela e gera mais agilidade.

Em comparação com as máquinas virtuais, os *containers* Linux revolucionaram a forma de desenvolvimento, gerenciamento e implantação das aplicações. Desse modo, de acordo com RedHat (2019a) a aplicação se mantém isolada, o que facilita muito o gerenciamento dos processos, permite portabilidade, escalabilidade e controle de versão, além de consumir menos recursos.

Figura 2 – Virtualização x Containers



Fonte: Imagem adaptada de https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container

2.1.2 Docker

O Docker é uma ferramenta *open source* que automatiza a implantação de aplicativos em *containers* (TURNBULL, 2017). Foi escrito pela equipe Docker e lançado sob a licença

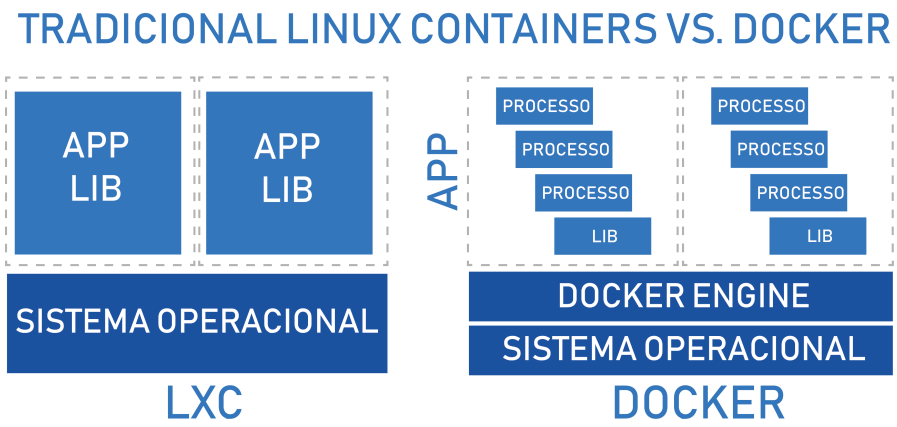
[1] kernel é o núcleo ou componente central de um sistema operacional. Ele funciona como uma via de comunicação entre aplicativos e o processamento real de dados feito a nível de hardware.

Apache 2.0. O Docker foi projetado para garantir um ambiente de desenvolvimento e execução de código de forma rápida e leve. Contendo um fluxo de trabalho eficiente para enviar o código de uma máquina qualquer para o ambiente de testes, e posteriormente para o ambiente de produção de forma simples.

Como o Docker não utiliza a camada de *hypervisor*, os *containers* são mais compactos e utilizam os recursos da forma mais eficiente. Tais características tornam o Docker uma abordagem de alto desempenho. Com isso, é possível reduzir o tempo de ciclo entre a gravação do código, teste, implementação e uso, possibilitando a fácil construção de aplicações portáteis de forma colaborativa.

A tecnologia utilizada pelo Docker não é a mesma dos *containers* tradicionais, apesar de ter sido bastante utilizada como uma virtualização mais leve no começo do desenvolvimento do Docker. Nos *containers* Docker foi adicionada uma camada entre as aplicações e o sistema operacional, como pode ser visto na Figura 3. Essa camada tem o objetivo de facilitar o processo de construção e criação dos *containers*, o que possibilita fazer o envio das imagens e um controle de versão de forma simplificada (REDHAT, 2019b).

Figura 3 – Linux Containers (LXC) x Docker



Fonte: Imagem adaptada de https://www.redhat.com/pt-br/topics/containers/what-is-docker

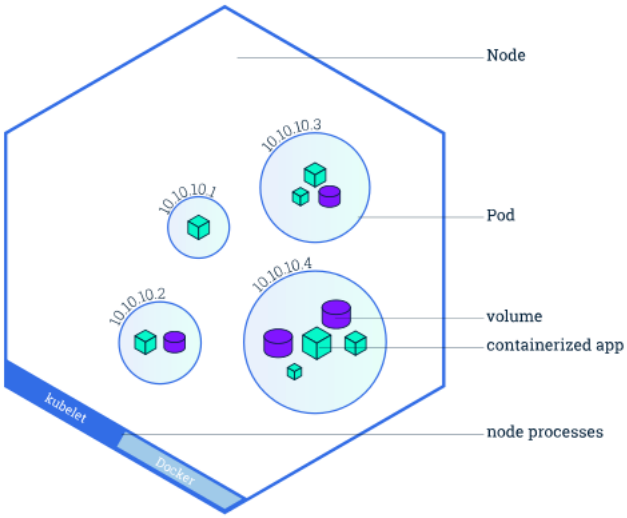
Arquiteturas orientadas a microsserviços são fortemente incentivadas pelo Docker. O recomendado é que cada *container* execute um único processo, serviço ou aplicação, que podem ser representados por uma série de *containers* interconectados, caracterizando aplicações distribuídas.

2.2 Kubernetes

Segundo RedHat (2019c), o Kubernetes, também conhecido como K8s, é um sistema de orquestração de *containers open source* que automatiza a implantação, o

dimensionamento e a gestão de aplicações em *containers*. Ele foi originalmente projetado pelo Google e agora é mantido pela *Cloud Native Computing Foundation* [2]. Um *cluster* de Kubernetes é composto por *nodes*, que são máquinas (físicas ou virtuais) que tem como principal função executar os *pods*. De acordo com Sayfan (2017), "os nós são abelhas operárias do Kubernetes que suportam todo o trabalho pesado". A estrutura de um *node* pode ser vista através da Figura 4.

Figura 4 – Estrutura de um Node

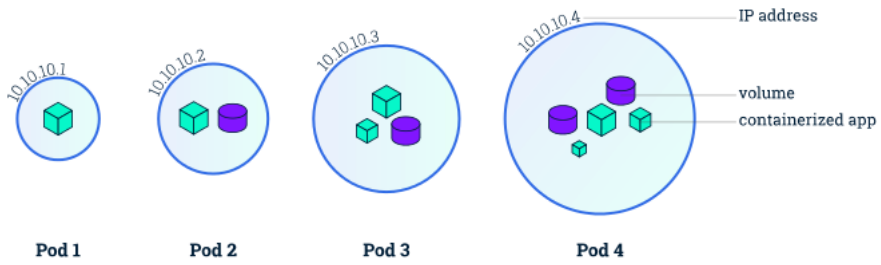


Fonte: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/node-overview

Os *pods* são a menor unidade de trabalho dentro do Kubernetes, podendo, cada um deles, ser composto por um ou mais *containers*. Os *pods* são sempre executados no mesmo node; por conta disso, todos os *containers* de um *pod* possuirão o mesmo endereço IP e porta que o próprio *pod*, com o intuito de realizar a comunicação entre eles via rede. Além disso, os *containers* que compõem o *pod* possuem armazenamento compartilhado com o *node* em que estão hospedados. Assim, os *pods* são entidades efêmeras, podem ser descartados e substituídos a qualquer momento (SAYFAN, 2017). A estrutura de um *pod* pode ser vista através da Figura 5.

[2] A Cloud Native Computing Foundation é atual mantenedora do projeto Kubernetes (https://www.cncf.io).

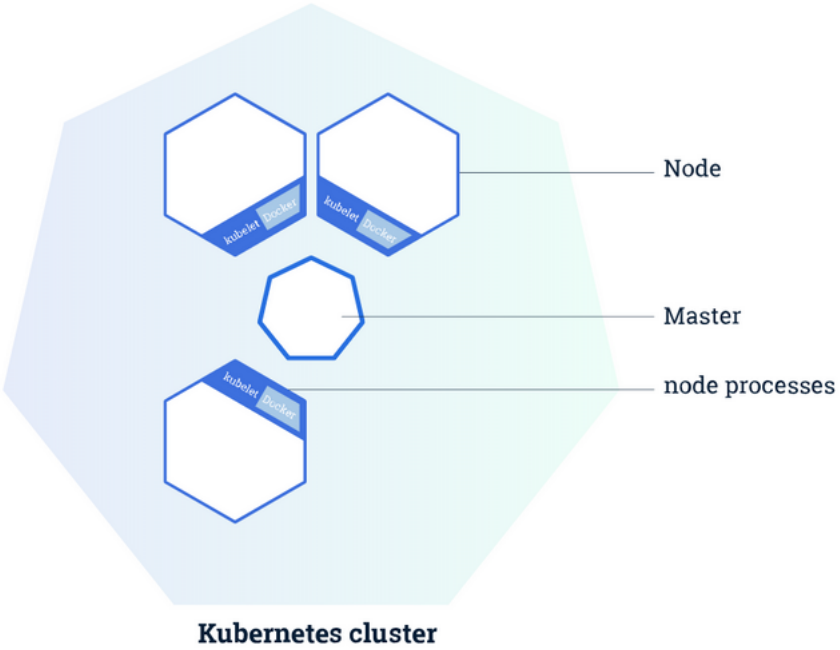
Figura 5 – Estrutura de Pod



Fonte: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/pods-overview

Um dos componentes fundamentais do Kubernetes é o Master, que é o responsável pela manipulação de todos os eventos que ocorrem dentro do Kubernetes e pelo agendamento global de todos os *pods* a nível do *cluster*. Para garantir o funcionamento do Master, muitas vezes, é utilizado um recurso de réplicas, para que o Master possa ser facilmente substituído caso algum problema de funcionamento ocorra, garantindo, então, o bom funcionamento do *cluster*. A estrutura de um *cluster* de Kubernetes pode ser vista através da Figura 6.

Figura 6 – Estrutura de um Cluster de Kubernetes

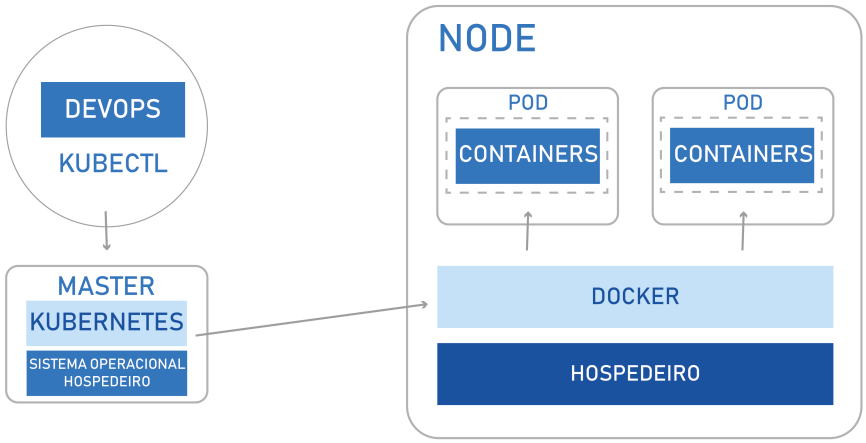


Fonte: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/cluster-diagram

Posto isso, é recomendado que apenas uma aplicação seja executada por *container* para que a transparência seja mantida. No que concerne à infraestrutura, implantar *containers* de forma individual implica em facilidade de uso e maior eficiência, visto que eles podem ser versionados, reconstruídos e reimplantados de forma independente.

O K8s se comunica com os *pods*, a máquina Master recebe os comandos e passa as instruções aos *nodes* que estão aptos a resolver as tarefas solicitadas. Uma vez que isso é feito, os recursos são alocados e atribuídos aos *pods* para que as requisições possam ser executadas, como mostra a Figura 7.

Figura 7 – Comunicação do Kubernetes com os Pods



Fonte: Imagem adaptada de https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes

2.3 *Cloud Computing*

No início, os *datacenters* em nuvem usavam virtualização como meio de consolidação de softwares e serviços nos servidores. O objetivo era reduzir a subutilização de recursos e aumentar a capacidade de gerenciamento, visto que, assim, era possível fazer o compartilhamento de hardware comum. O problema dessa estratégia era que, embora várias máquinas virtuais pudessem ser executadas em um mesmo servidor, cada VM executava uma cópia completa de um sistema operacional, o que gerava um consumo grande de recursos.

Os *containers* vieram como uma alternativa para solucionar esse problema. Como o armazenamento de todos os componentes necessários para a execução de um software específico é feito utilizando um conjunto mínimo de um sistema operacional Linux, os *containers* usam menos recursos que as máquinas virtuais e, além disso, os novos recursos podem ser provisionados de forma mais rápida do que nas VMs (LYNN et al., 2017).

Segundo a Amazon (2019), "a computação em nuvem é a entrega sob demanda de

poder computacional, armazenamento de banco de dados, aplicativos e outros recursos de TI pela Internet com uma definição de preço conforme o uso". Com a computação em nuvem, os recursos são disponibilizados com muita agilidade, no qual centenas de servidores e *clusters* podem ser implantados em questão de minutos em qualquer lugar do mundo. Hoje a computação em nuvem já é amplamente aceita como o paradigma dominante na computação. Segundo Gens (2016), se prevê que, até 2020, 67% de toda a infraestrutura de TI corporativa e gastos com software estejam baseados em ofertas baseadas em nuvem.

A escalabilidade é um dos pontos mais relevantes em relação à computação em nuvem, visto que ela permite que os recursos fornecidos possam ser aumentados ou diminuídos conforme as requisições aconteçam, o que é uma vantagem muito grande em relação à *datacenters* ou servidores físicos. Esse fator impacta diretamente na economia de custos, pois somente os recursos utilizados, pelo tempo que assim forem, serão cobrados. Outro fator de grande importância que deve ser levado em conta, é a segurança em relação a catástrofes naturais e manutenção dos dados, que passa a ser de responsabilidade do provedor de nuvem contratado, deixando os desenvolvedores focados no desenvolvimento, sem precisar se preocupar com a infraestrutura em si.

2.3.1 Tipos de computação em nuvem

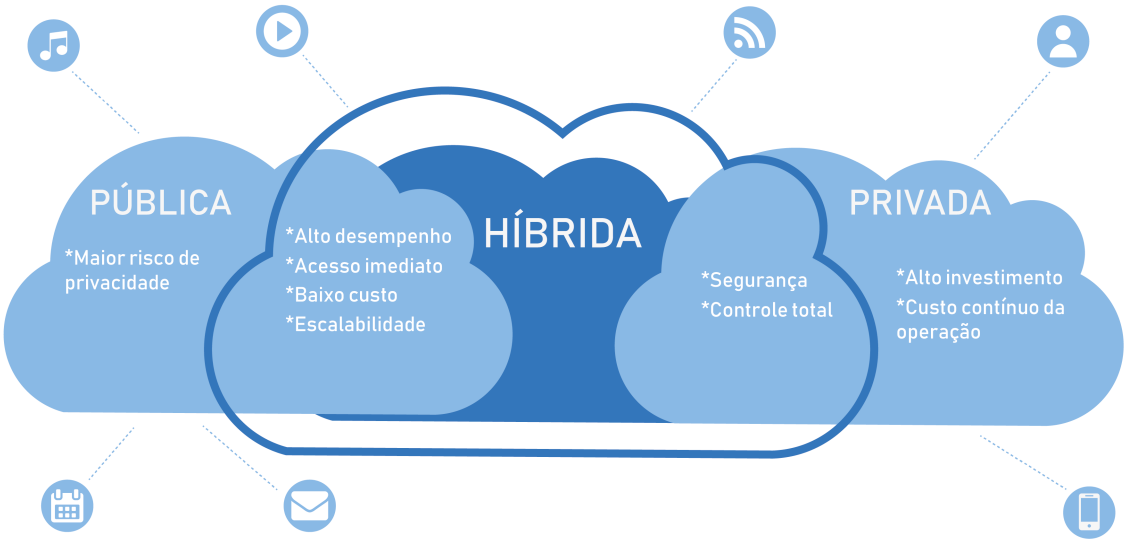
É muito importante determinar qual o tipo de nuvem ideal a ser implementada, dado que nem todas são iguais e não existe uma solução ideal para todos os cenários. Segundo a Microsoft (2019), existem três tipos de serviços em nuvem que podem ser implementados: a nuvem pública, a nuvem privada e a nuvem híbrida.

Na nuvem pública, toda a infraestrutura (hardware, software e suporte) é mantida pelo provedor de nuvem, os serviços são vinculados por uma conta, acessados e gerenciados através de um navegador web qualquer.

Já em uma nuvem privada, todos os recursos computacionais e de infraestrutura são exclusivos de uma única empresa ou organização. Essa nuvem pode estar em um *datacenter* da própria empresa, de forma física ou na Internet, sendo mantida por um provedor de nuvem.

E em uma nuvem híbrida, as tecnologias de nuvens públicas e privadas são combinadas, o que permite o compartilhamento de recursos, dados e aplicações entre elas, como é demonstrado na Figura 8. Essa tecnologia permite uma otimização da infraestrutura, flexibilidade, mais possibilidades de implantação e segurança (SCURRA, 2017).

Figura 8 – Tipos de computação em nuvem

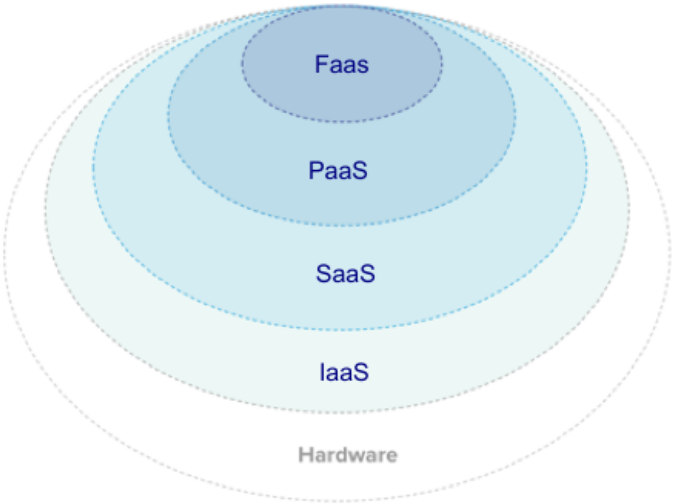


Fonte: Imagem adaptada de http://www.scurra.com.br

2.3.2 Tipos de serviços de nuvem

Com toda a evolução proporcionada pela computação em nuvem, novos tipos de serviços e formas de se trabalhar com TI acabaram surgindo. Esses modelos foram subdivididos em algumas categorias para melhor atender as demandas do mercado atual. Uma das formas de se perceber tais divisões podem ser vistas na Figura 9.

Figura 9 – Abstração de tipos de serviços de nuvem



Fonte: Imagem adaptada de https://content.pivotal.io

Uma dessas formas é a *Infraestructure as a Service* (IaaS) que permite que todos os recursos sejam gerenciados de forma instantânea através da internet, tudo isso de acordo

com a demanda exigida, podendo sofrer alterações de aumentos ou reduções de forma rápida e simples. Com a IaaS não existe a necessidade de gerenciar a complexidade dos recursos físicos, sendo possível, assim, adquirir somente o que é necessário, o que reduz consideravelmente os custos relacionados à infraestrutura (MICROSOFT, 2019).

Em uma segunda camada, tem-se o *Software as a Service* (SaaS) que possibilita entregar aplicações baseadas em nuvem, sob demanda pela internet, com toda a infraestrutura, manutenções e atualizações sendo mantidas pelos provedores de nuvem, tudo disponível para acesso através de um navegador *web* (MICROSOFT, 2019).

Já a *Platform as a Service* (PaaS) oferece ao usuário um ambiente completo de desenvolvimento, teste, entrega e gerenciamento de softwares sob demanda. Com a utilização da plataforma como serviço, os desenvolvedores não precisem se preocupar com configuração de ambiente e gerenciamento da infraestrutura, permitindo que o foco seja mantido somente no desenvolvimento em si (MICROSOFT, 2019).

Ainda, o *Function as a Service* (FaaS) também conhecido como *serverless computing*, veio para mudar a formar tradicional de desenvolvimento de software. Ao se utilizar aplicações nativas para *cloud*, a execução acontece sem que o usuário precise gerenciar ou provisionar o servidor e os recursos computacionais necessários (MICROSOFT, 2019).

Com o uso da computação em nuvem, os recursos de hardware passam a ser facilmente gerenciados, podendo ser contratados de acordo com a demanda necessária, o que gera uma grande economia de tempo e dinheiro.

2.4 Serverless Computing

A computação sem servidor pode também ser definida como uma arquitetura de software em que um aplicativo é decomposto em gatilhos (eventos) e ações (funções), por meio de uma plataforma que fornece um ambiente de hospedagem e execução contínua (GLIKSON; NASTIC; DUSTDAR, 2017). Apesar de uma maior eficiência e velocidade de provisionamento introduzidas pela virtualização e conteinerização, segundo Lynn et al. (2017), outras melhorias são limitadas pelo gerenciamento de recursos de infraestruturas subjacentes, no caso, os servidores.

A computação sem servidor possibilita um modelo de computação efetivo, em que todos os recursos, incluindo hardware, sistemas operacionais e ambientes de tempo de execução são todos agrupados. O termo "sem servidor"passa uma ideia errônea da inexistência de servidores; no entanto, existe uma infraestrutura e todo um ambiente de execução, que está totalmente provisionado, gerido e mantido pelo provedor de nuvem. É garantida, dessa maneira, uma plataforma escalável de forma automática e tolerante a falhas.

Na prática, isso é feito com aplicações preparadas para serem iniciadas ou desligadas a qualquer momento, as aplicações não guardam estados (*stateless*) e, preferencialmente, funcionam também de forma paralela, podendo executar múltiplas instâncias da mesma aplicação sem interferência no comportamento, com propósito único, em que seus processos de execução são realizados apenas sob demanda, geralmente através de uma API, sem consumir nenhum recurso até o ponto de execução, ou seja, sem gerar nenhum gasto até que sejam de fato executadas (KOLLER; WILLIAMS, 2017).

2.5 Knative

No Google Cloud Next 2018 [3], o Knative foi anunciado como "uma plataforma baseada em Kubernetes para *build*, *deploy* e gerenciamento de *serverless workloads* modernos". O *framework*[4] de código aberto busca codificar as melhores práticas de *build, serving requests*, e eventos. O Knative foi desenvolvido pelo Google em parceria com a Pivotal, IBM, Red Hat e a SAP (BRYANT, 2018).

O Knative (pronuncia-kay-nay-tiv) estende o Kubernetes para oferecer um conjunto de componentes de *middleware* fundamentais para construção de aplicativos modernos em *containers*, que podem ser executados em qualquer lugar, na nuvem ou, até mesmo, em um centro de dados de terceiros (KNATIVE, 2019d).

O Knative pode ser definido como um conjunto de abstrações construídas sobre o Kubernetes de modo a esconder detalhes complexos, fornecendo um *framework* que permite que os desenvolvedores possam focar no que realmente importa. O Knative permite que os desenvolvedores aproveitem o poder do Kubernetes, pois oferece um conjunto de componentes de código aberto que permite o desenvolvimento de aplicações *serverless*, que podem ser transferidas entre provedores de nuvem de forma muito simples, utilizando a tecnologia de *containers*. As aplicações que utilizam a tecnologia fornecida pelo Knative são totalmente portáteis, permitindo que sejam executadas de forma local ou em qualquer provedor de nuvem (BURT, 2018).

Através da junção do Kubernetes e Knative, é construída uma plataforma com a capacidade de executar aplicações *serverless*, com monitoramento integrado, permitindo escalar a aplicação para zero, a partir do momento que tal carga de trabalho não recebe mais requisições, o que ainda não era possível de ser feito.

Ainda, o Knative concentra-se em tarefas comuns de construção e execução de aplicações *cloud native*, na orquestração de *builds* de código para *containers*, vinculação de serviços a ecossistemas de eventos, gerenciamento e roteamento de tráfego de rede durante

[3] Maior evento internacional do Google, em sua edição do ano de 2018.

[4] Um framework é uma abstração de códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Fonte: https://pt.wikipedia.org/wiki/Framework

o *deploy*[5] e escalabilidade automática das cargas de trabalho. O *framework* disponibiliza uma padronização para se fazer *deploy* de qualquer aplicação, mesmo ela sendo tradicional, ou um *container* (MANOR, 2018).

A construção do Knative utiliza o Kubernetes e o Istio, como pode ser visto na Figura 10. O Istio é uma plataforma aberta projetada para conectar e proteger micro-serviços, através de um plano de controle que utiliza *Service mesh*[6] para Envoy Proxy [7], que permite que os desenvolvedores, administradores e provedores de plataforma que interagem com o *framework* possam ser contabilizados (BRYANT, 2018).

Figura 10 – Interação com Knative

Fonte: https://knative.dev/docs/

Segundo (BRYANT, 2018), o Knative disponibiliza para os desenvolvedores os seguintes componentes:

Build - Orquestração de *builds* de fontes-para-*container*

Eventing - Gerenciamento e entrega de eventos

Serving - Computação *request-driven* que pode escalar para zero

[5] Fazer um deploy corresponde a fase de implantação de um software no ambiente de produção.
[6] Service Mesh é uma rede de microserviços que oferece de forma consistente: segurança, service discovery, tracing de aplicações, monitoramento e tolerança a falhas, sem a necessidade de um recurso compartilhado.
[7] O Envoy é um proxy, um serviço de código aberto, projetado para aplicativos nativos da nuvem.

O componente de *build* do Knative é uma extensão do Kubernetes e utiliza primitivas existentes para permitir a execução de *builds* de *containers* no *cluster* a partir do código fonte. O objetivo é utilizar os recursos nativos do Kubernetes para obter código fonte de um repositório, fazer o *build* dentro de uma imagem de *container* e, então, executar esta imagem. Porém, a documentação cita que o usuário final do *framework* ainda é responsável por desenvolver os componentes correspondentes que executam a maioria destas funções. (BRYANT, 2018)

2.6 Trabalhos Relacionados

Kaviani, Kalinin e Maximilien (2019) ressaltam a importância de *serverless* como algo que diminui a complexidade de gerenciamento e escala de infraestrutura e em geral resultam em códigos mais simples, serviços mais baratos e alta disponibilidade. Como exemplo, há relatos de que a flexibilidade e simplicidade oferecida pela implantação de *serverless* permite que engenheiros desenvolvam dez vezes mais funções (aplicações totalmente serverless) do que microsserviços.

Estes mesmos autores analisam a relevância do Knative em comparação a outras plataformas capazes de prover um serviço de computação *serverless*. Essa comparação é feita principalmente baseando-se em suas APIs e protocolos de funcionamento como a forma que a plataforma permite que as aplicações se comuniquem (e.g.: protocolos HTTP/1 e HTTP/2), como elas devem ser conteinerizadas, e os modelos de execução e concorrência suportados. Por exemplo um ponto positivo do Knative é suportar ambos protocolos HTTP de modo que uma aplicação que utiliza funcionalidades desses protocolos pode rodar sem necessidade de alterações no Knative; em contrapartida, das plataformas comparadas, Knative é o único que ainda não suporta execução assíncrona. Apesar disso, como o Knative é desenvolvido sobre o Kubernetes se propõe a um modelo declarativo de configuração que é de fácil utilização. Dado isso, Kaviani, Kalinin e Maximilien (2019) concluem que "estes pontos em comum implicam que o Knative pode ser um bom candidato a seguir um caminho de sucesso similar ao do Kubernetes".

Levando em consideração todos os benefícios e pontos de atenção das plataformas *serverless* o Knative aparece como a plataforma mais promissora, pois similar ao Kubernetes ela é *open source*, extensível e flexível. Por conta disso os desenvolvedores não ficam reféns de nenhum provedor de computação em nuvem, diferente do que acontece ao se usar serviços proprietários fornecidos pelos provedores, o que é um dos principais pontos de preocupação encarados pelo desenvolvedores quando precisam tomar a decisão de mover seus códigos para uma plataforma *serverless*.

Por fim, Kaviani, Kalinin e Maximilien (2019) pontuam que "como para plataformas baseadas em *containers*, plataformas *serverless* experimentarão um ciclo de amadureci-

mento que irá resultar em uma camada *open source* comum como a base para futuras ofertas no mercado". Acreditando que as diferenças existentes nas plataformas analisadas são de fato boas pois irão permitir inovação na área de forma a consolidar produtos maduros.

Uma outra análise foi proposta por Silva e Carvalho (2019) com o objetivo de investigar o tempo de inicialização de aplicações inativas em um ambiente *serverless*, chamado de *coldstart*, onde foi feita a análise do impacto do tempo de resposta das aplicações. No estudo foram feitos experimentos que mediram esses impactos em um provedor de FaaS com o objetivo de responder os seguintes questionamentos:

Qual o tempo de ociosidade necessário para que a função seja desativada do provedor, o que resultaria no *coldstart* ao surgirem novas requisições?

Qual o tempo que será adicionado (*overhead*) por conta do *coldstart* no tempo de resposta das requisições comparando com as requisições que não sofreram *coldstart*?

Qual o impacto gerado por esse *overhead* do *coldstart*, relacionado a quantidade de memória alocada e qual o intervalo necessário para que o *coldstart* ocorra.

Com os resultados obtidos Silva e Carvalho (2019) concluíram que: “se as aplicações se mantém ativas e raramente apresentam longos períodos de atividade, a facilidade de uso do modelo de *serverless computing* pode ser um grande atrativo para os usuários executarem e escalarem suas aplicações sem precisar gerenciar servidores.” Entretanto se as aplicações precisarem responder com tempos menores que 1 segundo, o *overhead* imposto pelo *coldstart* pode afetar consideravelmente o desempenho da aplicação, sendo mais interessante utilizar um serviço tradicional baseado em máquinas virtuais ou *containers*.

De forma similar esses tempos de *coldstart* e *overhead* também foram testados neste trabalho na forma de *warmup* do *pod* ser iniciado (*coldstart*) e na análise do tempo de resposta médio de resposta com e sem Knative (*overhead*).

3 Metodologia

Para o desenvolvimento do estudo, um *cluster* de Kubernetes criado na Google Cloud foi utilizado. O Google oferece uma ferramenta que permite realizar o gerenciamento de máquinas, *clusters* e aplicações, chamada Google Kubernetes Engine. Por meio dessa ferramenta, foi realizada a criação e configuração do *cluster*, a implantação do Knative e todo o gerenciamento dos recursos necessários para a realização do estudo.

É importante salientar que o foco deste trabalho foi baseado no componente *Serving* do Knative, que é responsável por fazer o *scale to zero* de aplicações, no qual as maiores inovações e benefícios na utilização do Knative podem ser encontrados.

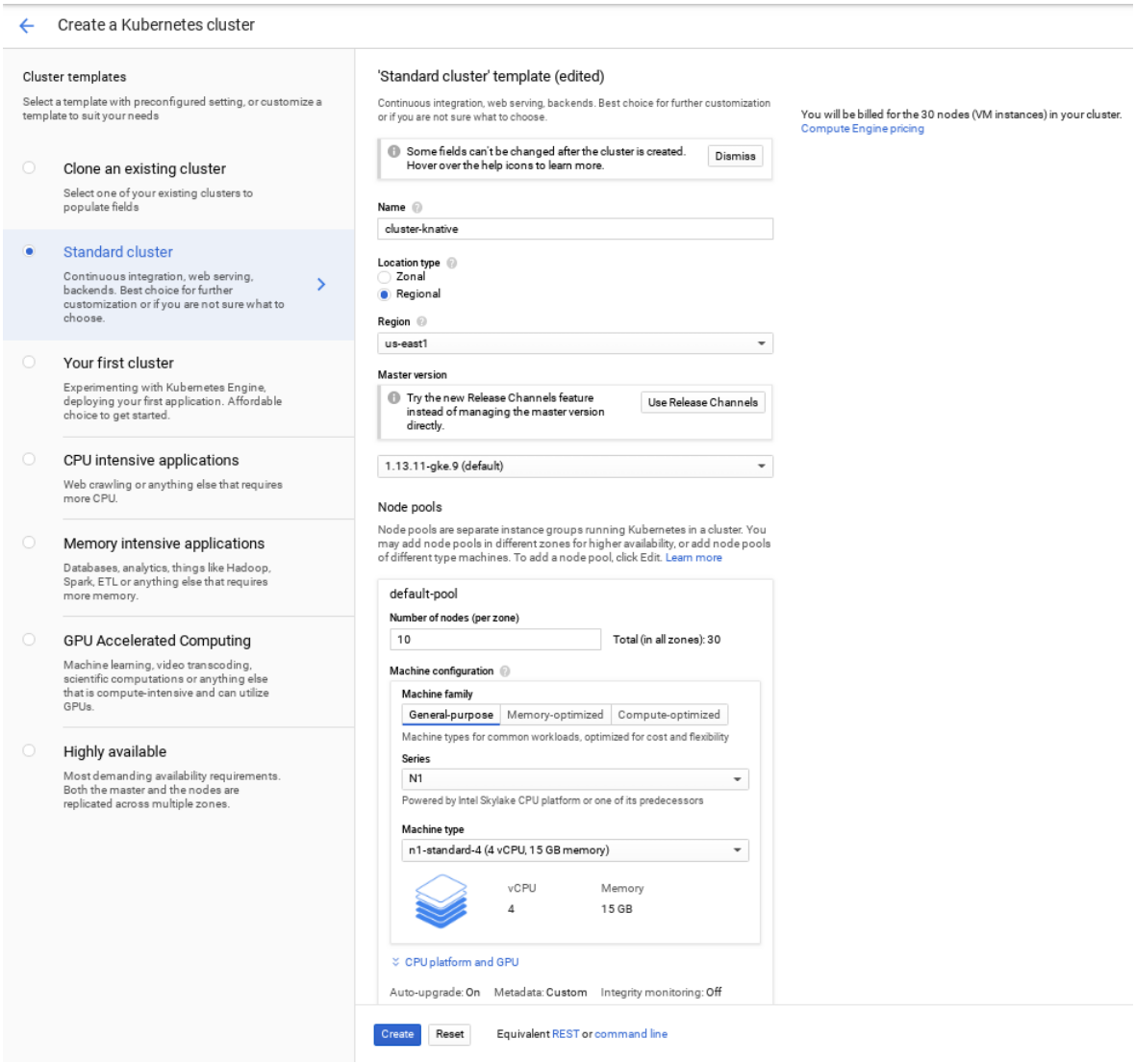
Para se obter uma prova de conceito do valor e potencial do Knative foram escolhidas duas aplicações específicas de código proprietário da Stilingue para serem adaptadas ao Knative afim de verificar o impacto do uso do Knative nas aplicações, reduzir gastos e otimizar o uso de recursos. No momento do desenvolvimento, as aplicações já estão em execução em produção respondendo a inúmeras requisições ao longo do dia, mas ficando ociosas durante a noite, desperdiçando assim os recursos que estão alocados para elas mas não estão sendo utilizados.

3.1 Google Kubernetes Engine

O *Google Kubernetes Engine* (GKE) oferece um ambiente que possibilita implantar, gerenciar e dimensionar aplicações em *containers* por meio da infraestrutura do Google. Seu ambiente é composto por várias máquinas agrupadas para formar *clusters* que são gerenciados pelo Kubernetes (GOOGLE, 2019a). Com o Kubernetes, torna-se possível executar tarefas de implantação, gerenciamento e fazer o monitoramento necessário para garantir a integridade das aplicações implantadas.

Um *cluster* de Kubernetes pode ser facilmente criado através da interface do GKE. Toda a configuração necessária como: nome, região do mundo em que o *cluster* será implantado, quantidade e tipos de *nodes*, além das configurações de rede necessárias para o seu funcionamento, podem ser feitas através do GKE, como pode ser visto através da Figura 11.

Figura 11 – Tela de configuração de um Cluster no GKE



Fonte: Google Kubernetes Engine, Google. Elaborado pelo autor.

A alocação de recursos (neste caso, memória RAM e CPU) pode ser feita adicionando *nodes* no *cluster* de Kubernetes. Esta alteração da quantidade de nodes também é conhecida como *scale*, o que consiste em alocar (aumentar ou diminuir) recursos do cluster. O scale também pode ser feito para *pods* da aplicação, a fim de aumentar o seu poder de processamento.

Uma configuração muito importante que precisa ser habilitada no *cluster* é a função de *autoscale*. Essa funcionalidade permite que o *cluster* consiga fazer *scale up* ou *scale down* horizontalmente dos *nodes* de acordo com a necessidade das aplicações controladas pelo Knative. Se a opção de *autoscale* não for habilitada, o Knative só conseguirá escalar na horizontal no nível de *pods* e não *nodes*. Nesse caso, o Knative vai gerenciar somente os recursos já disponíveis no *cluster*, o que não impede que novas aplicações sejam implantadas (desde que haja recurso disponível). Caso seja necessário adicionar ou remover *nodes*, esse

processo precisa ser feito manualmente.

Para a implantação do Knative, é necessário garantir que o *cluster* tenha recursos suficientes para hospedar o Knative e suas dependências, sendo assim a configuração mínima recomendada de cada *node* do *cluster* de Kubernetes é de 4 vCPU e 15 GB de memória RAM. Além disso, o *scale up* é limitado em até 10 nodes (KNATIVE, 2019c). Caso necessário essa configuração pode ser alterada posteriormente.

3.2 Aplicações adaptadas para a utilização do Knative

A adaptação das aplicações exige mudanças no código para se tornarem *stateless* de modo que não dependam de um armazenamento permanente para o seu funcionamento e possam ser livremente criadas e deletadas sem comprometer a qualidade e integridade das resposta dadas às requisições. É importante salientar que as aplicações foram escolhidas justamente por se adequarem aos pré-requisitos de um bom uso do Knative. As aplicações escolhidas são chamadas Message-receiver e Sam.

A Message-receiver tem a função de averiguar o funcionamento correto do ambiente de produção no que diz respeito a performance e comunicação entre aplicações, e por isso é executada sob demanda quando há a necessidade de fazer testes. Portanto, seu papel aqui é permitir uma análise do tempo de resposta, não fazendo sentido analisar seus custos visto que, uma vez que esses testes são realizados, ela é deletada do *cluster*.

Com relação a Sam tem-se uma aplicação muito ociosa nos períodos noturnos, onde sua demanda é menor. Sendo interessante assim que seu número de réplicas altere ao longo do dia de acordo com a necessidade de processamento, podendo até mesmo ser desligada nos momentos onde não há demanda. Para este caso, é interessante analisar os custos da aplicação porque o objetivo é otimizar o uso dos recursos, enquanto a análise do tempo de resposta não traz uma informação tão relevante pois seu funcionamento não é em tempo real.

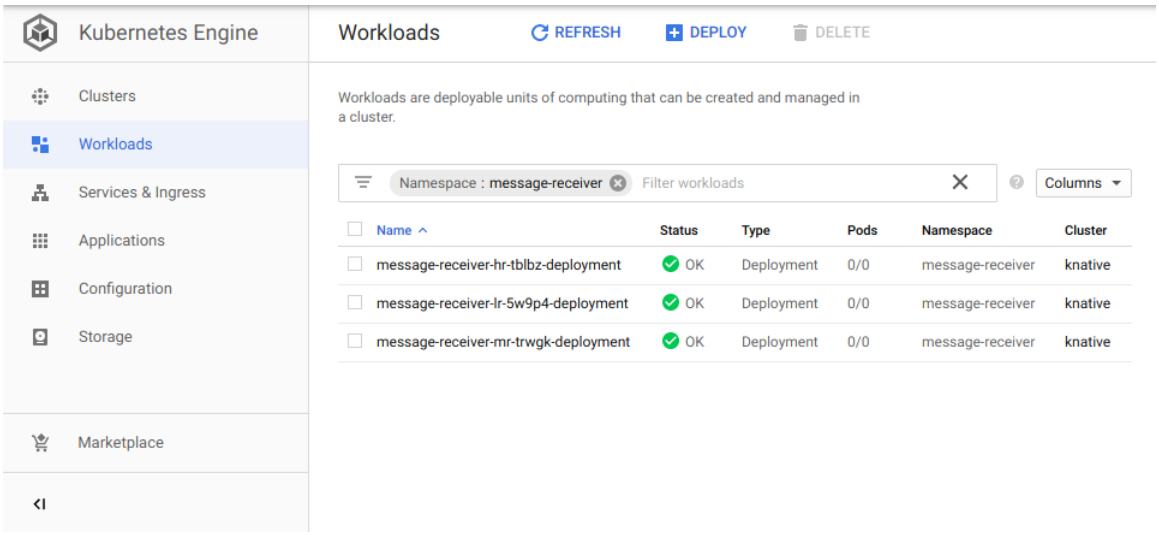
3.2.1 Aplicação Message-receiver

A Message-receiver é uma aplicação do tipo produtor-consumidor que processa dados em tempo real. Por ter essa característica, é uma aplicação muito utilizada para testes relacionados a tempo de resposta nos ambientes de *staging*[1] e produção em que o estudo foi realizado. Por conta disso, a aplicação Message-receiver foi escolhida para ser adaptada para o uso do Knative, na qual essas especificidades serão analisadas.

[1] Um ambiente de staging normalmente é utilizado para testar e revisar novas versões ou funcionalidades de uma aplicação antes de entrar em produção.

A Figura 12 ilustra a aplicação Message-receiver implantada com as três configurações de recursos diferentes. Pode-se ver, na coluna de *Workloads* do Google Kubernetes Engine, que a aplicação não possui *pods* ativos, visto que as aplicações não estão recebendo requisições no momento e, como consequência, foram escaladas para zero, ou seja, desligadas. Ao fazer isso, os recursos dessas aplicações não estão sendo consumidos (alocados) no *cluster*, o que os libera para que outras aplicações possam utilizá-los e, com isso, é possível gerar uma economia considerável de dinheiro.

Figura 12 – Deploys da aplicação Message-Receiver



Fonte: Google Kubernetes Engine, Google. Elaborado pelo autor.

3.2.2 Aplicação Sam

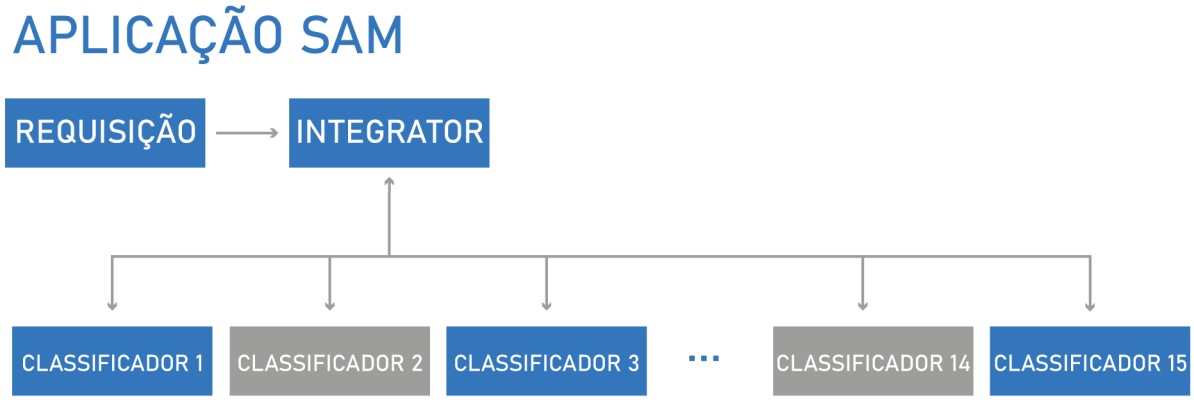
A Sam é uma aplicação de classificação de sentimentos composta por dois tipos de componentes. O primeiro componente, nomeado como *Integrator*, é responsável pelo roteamento dos dados de entrada para um modelo preditivo específico. O segundo componente é formado pelo conjunto de modelos preditivos distintos que recebem os dados a serem processados em *batch*.

No processamento em *batch*, as informações são coletadas ou recebidas, armazenadas e submetidas a um processamento posterior. No caso da aplicação Sam, cada tipo de *batch* é processado por um único modelo, por conta disso, é comum que nem todos estejam realizando processamento ao mesmo tempo, o que resulta em ociosidade e desperdício de recursos.

O primeiro componente, chamado de Integrator, fica sempre disponível, ou seja, seu *scale down* mínimo é definido como um, visto que ele é responsável por receber as requisições e encaminhar para a aplicação capaz de fazer o processamento de cada uma dessas requisições e gerar a resposta. O segundo componente é subdividido em 15 aplicações

classificadoras diferentes entre si, que ficam desligadas enquanto não recebem requisições e quando recebem sofrem *scale up* para responder à requisição feita. Esse funcionamento pode ser visto com mais detalhes através da Figura 13 onde os componentes representados pela cor azul encontram-se em execução e os componentes na cor cinza estão desligados pois não estão recebendo requisições no momento.

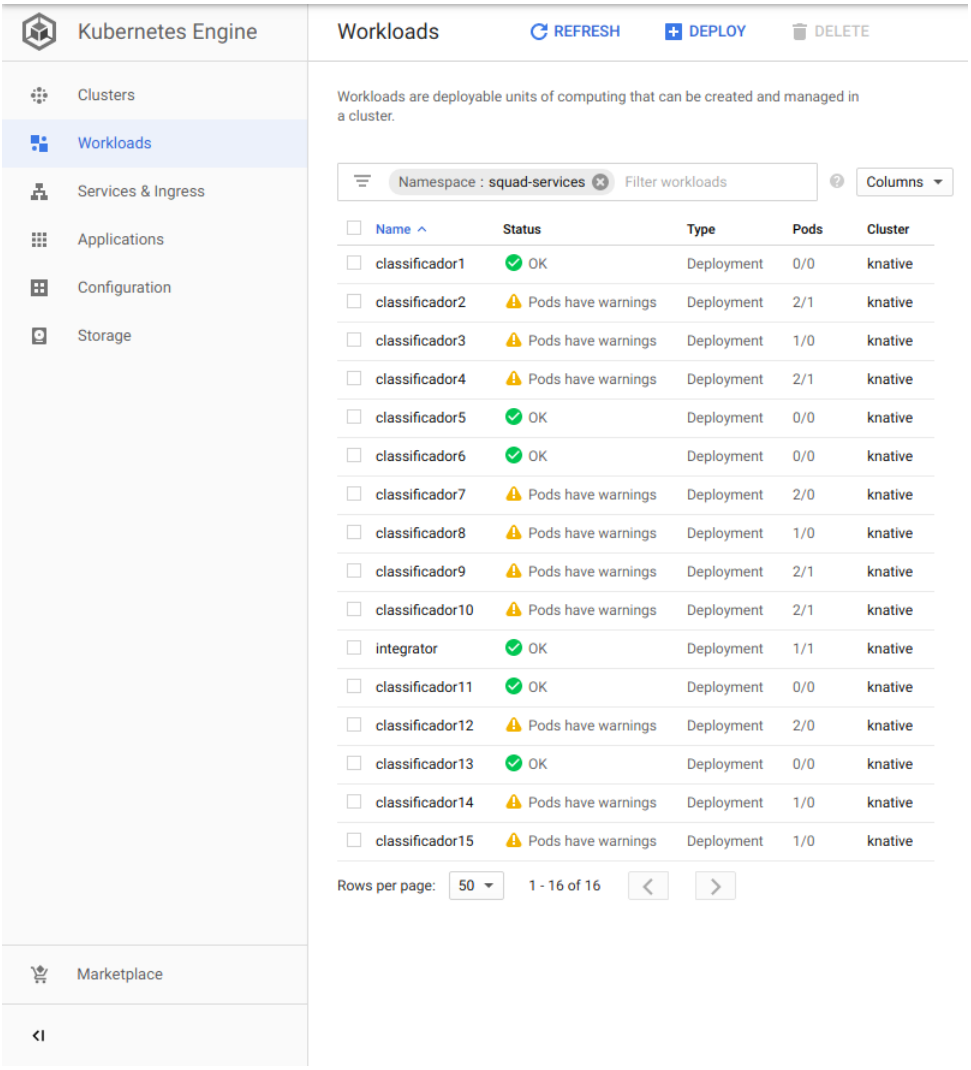
Figura 13 – Aplicação Sam



Fonte: Elaborada pelo autor.

Além disso, pode ser visto através do GKE, na Figura 14, que algumas aplicações apresentam a mensagem "*Pods have warnings*". Essa mensagem aparece pois alguns *pods* ainda estão em processo de desligamento (*scale to zero*) e por não estarem mais recebendo requisições; assim, quando esse processo é finalizado temos o *status* "0/0"na coluna *Pods*.

Figura 14 – Deploys da aplicação Sam



Fonte: Google Kubernetes Engine, Google. Elaborado pelo autor.

Sem o Knative, seria necessário que todas as 16 aplicações específicas estivessem sendo executadas simultaneamente o tempo integral (24 horas por dia, 7 dias por semana). Desta forma, o consumo da aplicação como um todo seria de 16 CPUs e 31 GB de RAM, uma vez que a aplicação responsável por encaminhar requisições utiliza 1 CPU e 1 GB de RAM e as outras 15, que fazem processamento, utilizam 1 CPU e 2 GB de RAM.

Com o Knative, torna-se possível não utilizar todos os recursos exigidos pelas aplicações (16 CPUs e 31 GB de RAM) ao mesmo tempo, dado que o Knative ficará responsável por fazer o *scale up* e *scale down* das aplicações quando for necessário, de acordo com as requisições feitas.

3.3 Implantação do Knative

A *cloud* do Google foi escolhida para a implantação do Knative por ser uma tecnologia que já tem integração nativa com o Kubernetes através do GKE, o que facilita a configuração do ambiente utilizado para os testes. É importante destacar que a versão do Knative utilizada foi a 0.8.0, por ser a versão mais nova no momento do estudo.

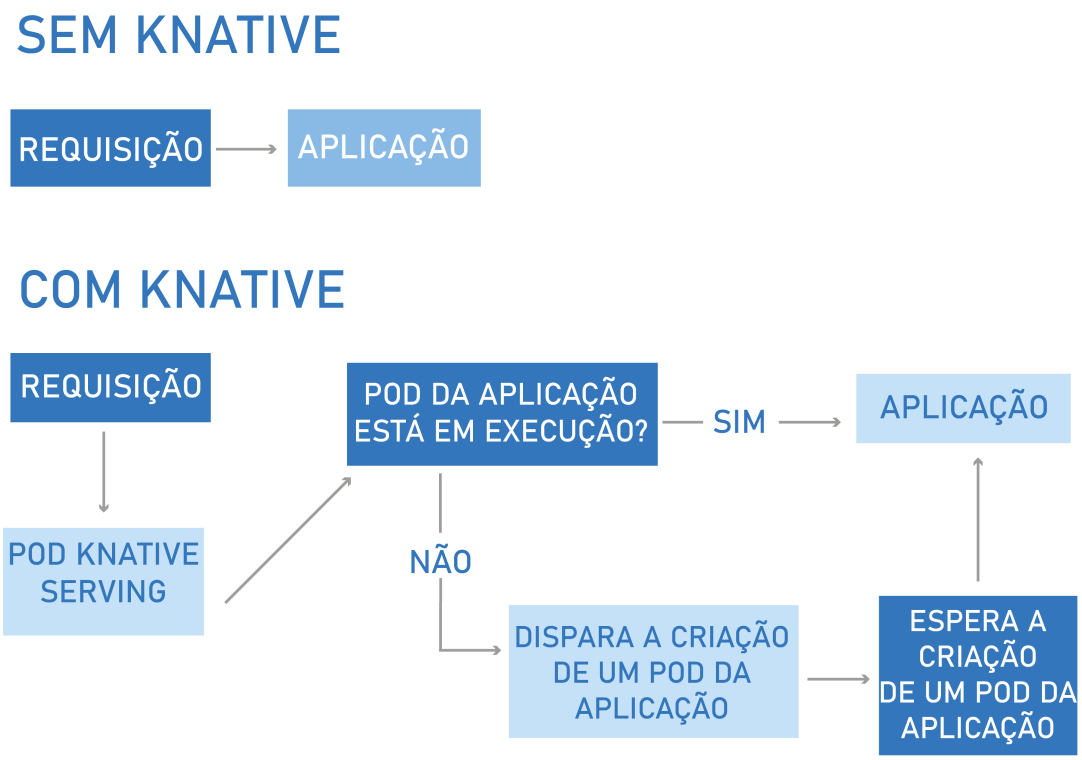
Desde o Knative v0.2, os *autoscalers* por revisão foram substituídos por um único *autoscaler* compartilhado. Esse é, por padrão, o *Knative Pod Autoscaler* (KPA), que fornece recursos rápidos de *autoscaling* com base nas requisições feitas (KNATIVE, 2019b). O funcionamento do *autoscale* para fazer *scale to zero* é definito através de três parâmetros, sendo eles: período de carência de escala para zero, janela estável e o período de rescisão.

O primeiro parâmetro é o período de carência de escala para zero (*scale-to-zero-grace-period*), que especifica o tempo que uma revisão inativa permanece em execução antes de ser redimensionada para zero e que, por padrão, leva 30 segundos. Como segundo parâmetro, tem-se a janela estável (*stable-window*) que corresponde ao tempo que o *autoscaler* espera para analisar o comportamento da revisão antes de decidir terminá-la. Essa análise é realizada sobre a quantidade de requisições recebidas durante essa janela de 60 segundos. Ou seja, o Knative analisa o comportamento da aplicação por 60 segundos e, se nesse tempo a aplicação não receber nenhuma requisição, ela é agendada para ser encerrada; sendo que ela será encerrada imediatamente se houverem 1 ou mais *pods* além dela ou, se ela for o único *pod* no momento, o *autoscaler* fará a contagem regressiva de 30s (*scale-to-zero-grace-period*) para fazer o *scale to zero*. E, por último, tem-se o período de rescisão, que é o encerramento total dos *pods* da aplicação, ou seja, é o tempo que o último *pod* leva para desligar após a conclusão da última solicitação. O período de término do último *pod* é igual à soma dos valores dos parâmetros *stable-window* e *scale-to-zero-grace-period*. Nesse caso, o período de rescisão seria de 90s; ambos os tempos podem ser configurados de acordo com a necessidade do funcionamento das aplicações.

Já o funcionamento do *autoscale* para fazer *scale up* é configurando através dos limites de solicitação simultânea, ou seja, são definidas quantas solicitações simultâneas são desejadas em um determinado momento (limite flexível). O valor padrão para o destino de simultaneidade especificado como 100, que é a configuração recomendada para o dimensionamento automático no Knative, (KNATIVE, 2019b).

A Figura 15 ilustra, em sua primeira parte, como seria o funcionamento de uma requisição sem a utilização do Knative, isto é, a requisição é feita e a aplicação precisa estar sempre disponível para responder as solicitações realizadas.

Figura 15 – Comparativo do funcionamento das requisições sem Knative e com Knative



Fonte: Imagem elaborada pelo autor.

Na segunda parte, pode-se ver como é o funcionamento das requisições nas aplicações adaptadas para utilizar o Knative. A requisição é feita e o componente *Serving* do Knative verifica se já existe alguma instância ativa, no caso, um ou mais *pods*, se sim, essa requisição chega na aplicação e é processada. Se não, o componente *Serving* faz o *scale up* de novos *pods*. Uma vez que os *pods* foram criados, a aplicação recebe a requisição e faz o processamento normalmente.

3.4 Dificuldades

No momento da redação desta monografia (2019), o Knative está completando 1 ano de criação. Deste modo, esta ferramenta está em pleno desenvolvimento, umas vez que recursos fundamentais para o seu funcionamento ainda não estavam disponíveis por padrão. Tal situação demandou um trabalho extra para que toda a configuração do Knative pudesse ser feita.

Para que as aplicações que utilizariam a tecnologia pudessem ser colocadas em produção foi necessário esperar o lançamento de novas versões e refazer todo o processo de configuração do Knative e das aplicações que seriam adaptadas; visto que foi levado

em consideração a segurança das aplicações e do próprio *cluster*, além, ainda, das suas disponibilidades.

4 Resultados

Esse capítulo tem como objetivo apresentar os resultados de duas análises realizadas em aplicações que estão em ambiente de produção, na empresa onde o estudo foi realizado, sendo uma análise de desempenho e outra de custos.

Para a análise de desempenho, o objetivo é medir o tempo de resposta e, para que essa análise seja realizada, a aplicação Message-receiver será utilizada. Para a análise de custos, foi realizado um comparativo entre os gastos gerados pela aplicação Sam configurada para utilizar o Knative e da mesma aplicação sem o seu uso.

4.1 Análise de desempenho

Analisar o desempenho das aplicações é de suma importância, uma vez que o Knative insere um componente entre a requisição a ser processada e a aplicação que realizará esse processamento. Com isso, espera-se avaliar se a alteração no fluxo de processamento das requisições, feita pelo Knative, apresenta algum impacto, no que corresponde ao tempo que a aplicação levará para responder tal requisição comparando-se ao resultado da mesma ação e aplicação sem utilizar o Knative.

Dois parâmetros serão avaliados para fazer a análise de desempenho : tempo de *warm up* e tempo de resposta. O tempo de *warm up* é o tempo necessário para a aplicação estar apta a responder as requisições que recebe de forma consistente. Deve-se ressaltar que, em casos onde o tempo de *warm up* não possa ser elevado, é possível deixar os *pods* com *scale down* mínimo de uma instância. Com isso, as requisições serão respondidas de forma mais rápida e podem ser armazenadas em um *buffer*[1], que será processado assim que os *pods* necessários entrem em execução.

Para verificar o tempo de resposta com a utilização do Knative, é preciso levar em consideração os três cenários possíveis:

Cenário 1 - O *Cluster* está com a aplicação em execução no momento da requisição: esse cenário implica que a aplicação está pronta para responder e espera-se um resultado semelhante ao tempo de resposta da aplicação sem Knative;

Cenário 2 - O *Cluster* tem recursos disponíveis, mas a aplicação não está em execução no momento da requisição: neste caso, é necessário que o Knative inicie um *pod* com a aplicação para que ela possa responder as requisições que estão chegando. Isso

[1] Região de memória física utilizada para armazenar temporariamente os dados que serão posteriormente processados

gera um pequeno tempo de *warm up*, que é o tempo necessário para escalar um novo *pod*, somando-se ao tempo de resposta;

Cenário 3 - O *Cluster* não tem recursos disponíveis e a aplicação não está em execução no momento da requisição: este é o cenário analisado que leva um maior tempo de inicialização. Nesse cenário, o *cluster* precisa realizar *scale up* de, pelo menos, um *node* e pelo menos um *pod*, o que gera um tempo um pouco maior de *warm up*; uma vez que essa nova máquina é adicionada ao *cluster* é feito *scale up* de um novo *pod* para responder as requisições. Isso gera um atraso em média de um a dois minutos.

Para a análise de desempenho, considerando o tempo de *warm up* somado ao tempo de resposta, utilizou-se da aplicação Message-receiver com três configurações de recursos diferentes: *low resources* (lr), *medium resources* (mr) e *high resources* (hr), que podem ser vistas na Tabela 1.

O *scale up* máximo das aplicações gerenciadas pelo Knative foi definido como cinco, o que faz toda a diferença no desempenho, visto que quanto mais aplicações em execução, mais rápido será o processamento, porém, mais recursos serão alocados.

Tabela 1 – Configurações de recursos da aplicação Message-receiver

Aplicação	CPU	RAM
message-receiver-lr	1	2 GB
message-receiver-mr	3	4 GB
message-receiver-hr	5	10 GB

Fonte: Elaborado pelo autor

A Message-receiver foi utilizada para os testes por ser uma aplicação mais sensível a atrasos no tempo de resposta e, portanto, é mais importante que nela o tempo seja semelhante ao uso convencional sem Knative, com a utilização do Knative em escala automática.

Acredita-se que, para testar uma hipótese bem fundamentada, seria importante a realização de vários testes com as mesmas condições e, por fim, analisar a média dos resultados obtidos. Porém, devido ao contexto do estudo, não é possível garantir que todos os testes aconteçam sob as mesmas condições; visto que a quantidades de *nodes* e de *pods* precisam ser iguais, o que é fundamental para a reprodução dos testes nos cenários definidos. Por isso, priorizou-se a simulação de um ambiente crítico, com uma quantidade elevada de requisições (1000) e a avaliação do comportamento do sistema sob este momento de *stress*.

Para a realização dos testes, a ferramenta de *benchmark*[2] Hey[3] foi utilizada, na qual foi definido que serão feitas 1.000 requisições, sendo 50 simultâneas. A escolha dos valores destes parâmetros foi realizada com base em testes preliminares. Uma vez que o teste for inicializado, o Knative controlará todo o *scale up* e *down* dos *pods* e *nodes*.

Para as aplicações configuradas com recursos diferentes sem a utilização do Knative, tem-se os resultados apresentados pela Tabela 2.

Tabela 2 – Tempo de resposta aplicações Message-receiver sem Knative

Aplicações	Requisições/s	Mais rápido(s)	Mais lento(s)	Média(S)	Total(s)
message-receiver-lr	306.3850	0.1352	0.3259	0.1591	3.2639
message-receiver-mr	294.3931	0.1366	0.4317	0.1630	3.3968
message-receiver-hr	298.6078	0.1364	0.4774	0.1619	3.3489

Fonte: Elaborado pelo autor

No momento do teste, o *Cluster* estava com sete *nodes* e cinco *pods* da aplicação em execução. É importante salientar que para esse resultado não existem cenários diferentes, visto que a aplicação fica sempre em execução no *Cluster*. Além disso, é possível que ocorram pequenas variações de tempo, por conta da rede, ou problemas externos. Entretanto, cuidados adicionais foram tomados para o que o ambiente de testes fosse o mais semelhante possível, na tentativa de assim, obter bons resultados.

4.1.1 Comparativo Cenário 1

Durante os testes, o *Cluster* estava com sete *nodes* e cinco *pods* da aplicação em execução, gerando os resultados que podem ser vistos na Tabela 3.

Tabela 3 – Tempo de resposta aplicações Message-receiver com Knative - Cenário 1

Aplicações	Requisições/s	Mais rápido(s)	Mais lento(s)	Média(S)	Total(s)
message-receiver-lr	249.7499	0.1382	0.7280	0.1897	4.0040
message-receiver-mr	293.8370	0.1364	0.6328	0.1674	3.4032
message-receiver-hr	288.8191	0.1380	0.6984	0.1688	3.4624

Fonte: Elaborado pelo autor

[2] Benchmark é o ato de executar um conjunto de programas ou outras operações, a fim de avaliar o desempenho relativo de um objeto, que normalmente consiste em executar uma série de testes padrões.
[3] O Hey é um pequeno programa que envia uma carga para um aplicativo da web.

Tabela 4 – Comparativo do tempo de resposta - Cenário 1

Aplicações	Média sem Knative (s)	Média com Knative (s)	Diferença entre os tempos de resposta (s)
message-receiver-lr	0.1591	0.1897	0.0306
message-receiver-mr	0.1630	0.1674	0.0044
message-receiver-hr	0.1619	0.1688	0.0069

Fonte: Elaborado pelo autor

Nesse cenário, o resultado obtido foi muito próximo ao das aplicações que não utilizam o Knative, como pode ser visto através da Tabela 4. Uma vez que as aplicações utilizando o Knative já estejam em execução seu funcionamento ocorre de maneira semelhante as aplicações que não estejam utilizando o Knative.

4.1.2 Comparativo Cenário 2

No decorrer dos testes, o *Cluster* também estava com sete *nodes*, porém, com zero *pods* da aplicação em execução, dado que a aplicação já tinha sofrido *scale to zero* pelo Knative e, por conta disso, não estava recebendo requisições. É necessário notar que é de suma importância que o número de *nodes* seja o mesmo, visto que esse fator pode influenciar diretamente nos resultados apresentados pela Tabela 5.

Tabela 5 – Tempo de resposta aplicações Message-receiver com Knative - Cenário 2

Aplicações	Requisições/s	Mais rápido(s)	Mais lento(s)	Média(S)	Total(s)
message-receiver-lr	38.6826	0.1377	1.5502	0.2047	25.8514
message-receiver-mr	53.5160	0.1378	15.3860	0.9262	18.6860
message-receiver-hr	49.3298	0.1390	14.5646	0.8856	20.2717

Fonte: Elaborado pelo autor

Tabela 6 – Comparativo do tempo de resposta - Cenário 2

Aplicações	Média sem Knative (s)	Média com Knative (s)	Diferença entre os tempos de resposta (s)
message-receiver-lr	0.1591	0.2047	0.0456
message-receiver-mr	0.1630	0.9262	0.7632
message-receiver-hr	0.1619	0.8856	0.7237

Fonte: Elaborado pelo autor

Neste cenário, é possível observar que o tempo de resposta foi maior. No entanto, a diferença de tempo é muito pequena, como pode ser visto na Tabela 6. Isso acontece porque o Knative precisa fazer *scale up* de *pods*, o que gera um tempo de *warm up*. É

importante lembrar que esse tempo só é somado nas primeiras requisições até que os *pods* sejam iniciados; uma vez em execução, a aplicação passa a responder normalmente com um tempo semelhante à aplicação sem a utilização do Knative. Também é possível observar que o número de requisições por segundo diminuiu, isso ocorreu por conta de existir o tempo de *warm up* para que a aplicação inicie e possa responder nesse cenário.

4.1.3 Comparativo Cenário 3

Ao longo dos testes, apenas dois *nodes* estavam em execução, o que já era esperado, pois o *cluster* estava a um tempo sem receber requisições e, como consequência, os *nodes* também sofreram *scale down*. Posto isto, é importante entender que *Clusters* utilizando o Knative nunca ficam com zero *nodes*, dado que algumas aplicações fundamentais para o funcionamento do Knative ficam sempre em execução.

Nesse cenário, é apresentado o maior tempo de resposta, como pode ser visto na Tabela 7, já que não existe recurso disponível no *cluster*. Por conta disso, o Knative vai precisar fazer *scale up* de *nodes*, antes de fazer *scale up* de *pods*, o que resulta no maior tempo de *warm up* e no pior cenário. Contudo, uma vez que os *nodes* e *pods* são iniciados (sete *nodes* e cinco *pods*), as aplicações passam a responder com um tempo semelhante as aplicações que não utilizam o Knative como pode ser visto na Tabela 8.

Tabela 7 – Tempo de resposta aplicações Message-receiver com Knative - Cenário 3

Aplicações	Requisições/s	Mais rápido(s)	Mais lento(s)	Média(S)	Total(s)
message-receiver-lr	34.6761	0.1417	4.4303	0.4487	28.8383
message-receiver-mr	11.0746	0.1397	5.9250	0.5447	90.2967
message-receiver-hr	9.8833	0.1389	18.9513	1.3076	101.1811

Fonte: Elaborado pelo autor

Tabela 8 – Comparativo do tempo de resposta - Cenário 3

Aplicações	Média sem Knative (s)	Média com Knative (s)	Diferença entre os tempos de resposta (s)
message-receiver-lr	0.1591	0.4487	0.2896
message-receiver-mr	0.1630	0.5447	0.3817
message-receiver-hr	0.1619	1.3076	1.1457

Fonte: Elaborado pelo autor

Esses resultados demonstram que aplicações que não precisam de resposta em tempo real, podem, perfeitamente, utilizar o Knative sem gerar grandes perdas de desempenho; visto que o maior tempo de *warm up* gerado pelo *scale up* de *nodes* não ocorre com frequência e, mesmo quando ocorre, fica em média entre um e dois minutos. Passado isso,

a aplicação utilizando Knative responderá normalmente com uma diferença irrelevante ao tempo de resposta das aplicações que não utilizam o Knative.

4.2 Análise de custos

As empresas (de todos os portes) buscam, de forma constante, a redução dos altos custos relacionados à infraestrutura. Desse modo, tecnologias que possam reduzir o consumo elevado de recursos, sem gerar impactos considerados negativos no desempenho das aplicações, são fundamentais.

Os custos das máquinas no GCP são estimados em dólares e o valor depende do tipo de máquina e da região do mundo onde ela está alocada. Esses custos foram obtidos utilizando a calculadora oficial de estimativa do Google, que leva em consideração as configurações e tempo de uso de um produto para estimar um valor. Para os testes em questão, o produto é *Compute Engine* que corresponde às instâncias de máquinas no GCP.

Além disso, descontos podem ser concedidos com a utilização (alocação) contínua das máquinas ao longo do mês. Esses descontos podem ser até 30% proporcional ao uso mensal (GOOGLE, 2019b) ou um desconto de máquinas preemptivas[4] que é aproximadamente 70%, de acordo com os valores apresentados na Tabela 10) (GOOGLE, 2019c). Outro ponto importante a ser mencionado é que o Google cobra o primeiro minuto inteiro, depois disso, são cobrados apenas os segundos em que os recursos alocados são utilizados.

No exemplo utilizado no estudo apresentado, a região escolhida foi Carolina do Sul, nos Estados Unidos da América (us-east1), por apresentar um custo-benefício melhor, levando-se em consideração as interações (requisições) vindas do Brasil. Apesar de existir a região de São Paulo, no Brasil (southamerica-east1), ela também ser oferecida pelo Google, a mesma apresenta custos adicionais que tornam as máquinas mais caras que em uma região fisicamente mais distante, como pode ser visto na Tabela 9.

Tabela 9 – Comparativo mensal de preços de máquinas por região no GCP

Tipo de máquina	Região	Custo mensal (USD)
n1-standard-4	Carolina do Sul (us-east1)	$97.09
n1-standard-4	São Paulo (southamerica-east1)	$154.12

Fonte: Google Cloud Platform Pricing Calculator, Google. Elaborado pelo autor.

O tipo da máquina escolhida foi a *n1-standard-4* por recomendação da documentação do Knative (KNATIVE, 2019c), porém, essa configuração pode ser alterada para

[4] As máquinas preemptivas podem ser destruídas a qualquer momento, uma vez que isso acontece, novas máquinas são criadas.

melhor se adaptar aos recursos exigidos pelas aplicações implantadas no *cluster*. Os valores e tipos de máquinas disponíveis na região *us-east1* podem ser vistos na Tabela 10.

Tabela 10 – Tipos e custos mensais de máquinas N1 no GCP

Tipo de máquina	CPUs virtuais	Memória	Preço (USD)	Preço preemptivo (USD)
n1-standard-1	1	3,75 GB	$24.2725	$7.30
n1-standard-2	2	7.5 GB	$48.5500	$14.60
n1-standard-4	4	15 GB	$97.0900	$29.20
n1-standard-8	8	30 GB	$194.1800	$58.40
n1-standard-16	16	60 GB	$388.3600	$116.80
n1-standard-32	32	120 GB	$776.7200	$233.60
n1-standard-64	64	240 GB	$1553.4400	$467.20
n1-standard-96	96	360 GB	$2330.1600	$700.80

Fonte: https://cloud.google.com/compute/all-pricing, acessada em 20/11/2019

Os tipos de máquinas também podem ser personalizados de acordo com a necessidade e especificidade dos recursos exigidos pelas aplicações implantadas no *cluster*. Para isso, existem máquinas focadas em RAM, máquinas focadas em CPU e máquinas personalizadas que atendem demandas específicas. Apesar de as máquinas personalizadas apresentarem um custo mais alto, como pode ser visto na Tabela 11, dependendo das aplicações do *cluster*, isso pode resultar em uma economia no final das contas. Isso acontece, pois, uma vez que as máquinas são específicas para as aplicações implantadas no *cluster*, seus recursos podem ser melhor aproveitados.

Tabela 11 – Custos das máquinas N1 personalizadas no GCP

Item	Preço sob demanda	Preço preemptivo	Preço de compromisso por um ano	Preço de compromisso por três anos
vCPUs personalizadas	$16.95/vCPU ao mês	$5.10/vCPU ao mês	$14.53795/vCPU ao mês	$10.38425/vCPU ao mês
Memória personalizada	$2.27/GB ao mês	$0.68/GB ao mês	$1.94837/GB ao mês	$1.39211/GB ao mês

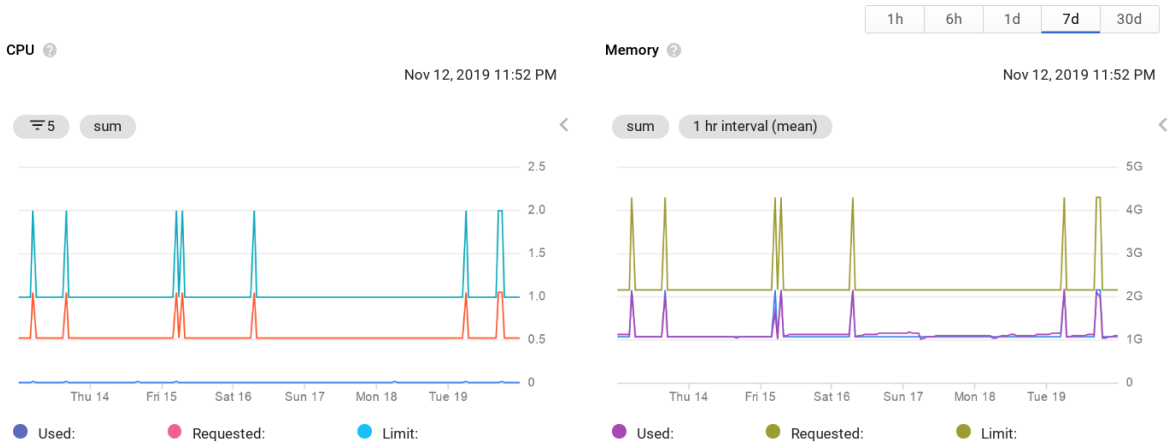
Fonte: https://cloud.google.com/compute/all-pricing, acessada em 20/11/2019

Para fazer a análise de custos, foi utilizada a aplicação Sam, que já estava implantada em ambiente de produção sem a utilização do Knative. A escolha se deu, visto que a aplicação Sam possui características que são exigidas (a aplicação ser *cloud native* e *stateless*) para funcionar com o Knative. Isso se dá pelo fato da aplicação ser composta por outras aplicações menores que não necessariamente precisam estar em execução simultaneamente. Neste ponto, o Knative gerencia quando uma dessas aplicação fará *scale*

up ou *scale down*, evitando desperdício de recursos e dinheiro. Tudo isso torna a aplicação Sam ideal para a utilização do Knative.

Na Figura 16, pode ser visto o funcionamento de um dos classificadores sem o Knative no período de sete dias. Neste período, a aplicação ficou em execução durante todo o tempo. Onde o eixo Y mostra a quantidade de recursos consumidos e o eixo X mostra como foi esse consumo ao longo do tempo.

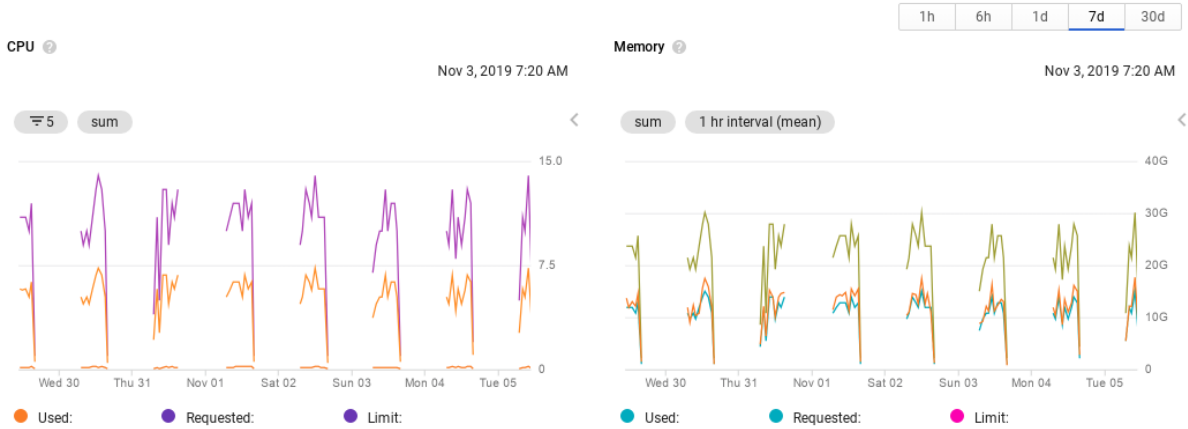
Figura 16 – Monitoramento de um dos classificadores sem utilizar o Knative



Fonte: Google Kubernetes Engine, Google. Elaborado pelo autor.

Já na Figura 17, percebe-se o funcionamento de um dos classificadores com o Knative, também no período de sete dias. Neste caso, é possível ver valores no gráfico que indicam períodos em que a aplicação não estava em execução. Ambos os casos foram obtidos através do monitoramento nativo do GKE.

Figura 17 – Monitoramento de um dos classificadores utilizando o Knative

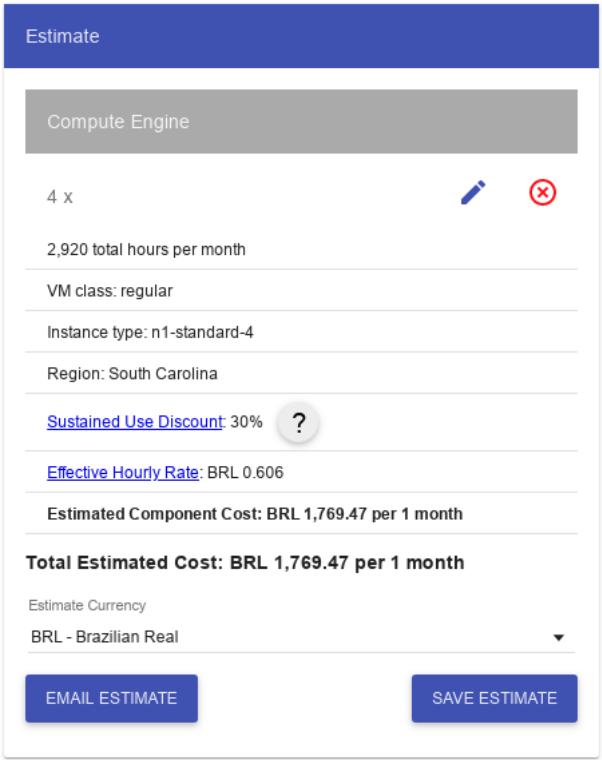


Fonte: Google Kubernetes Engine, Google. Elaborado pelo autor.

A aplicação Sam, sem a utilização do Knative, seria executada com todos os recursos

exigidos 24 horas por dia, 7 dias por semana, o que geraria um custo que pode ser visto através da Figura 18.

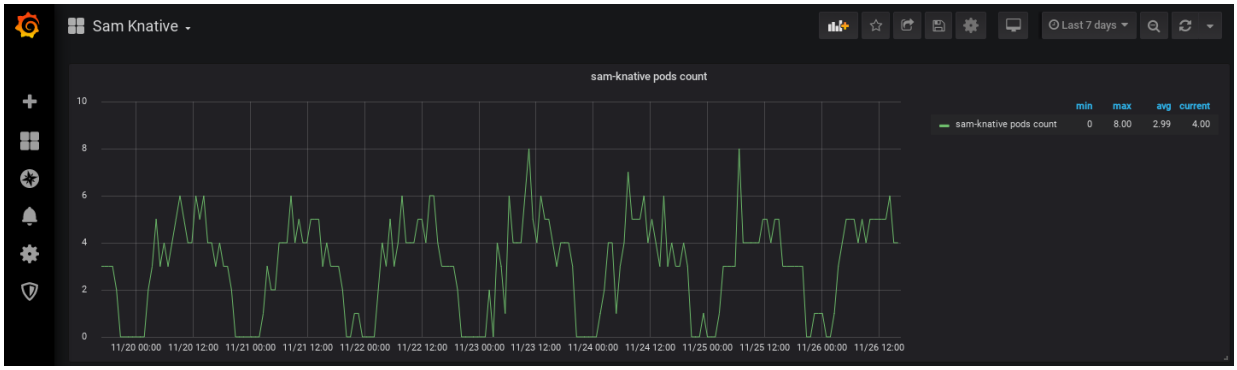
Figura 18 – Custo total da aplicação Sam sem o Knative



Fonte: Google Cloud Platform Pricing Calculator, Google. Elaborado pelo autor.

Por meio do Grafana[5], foi possível obter a média de *pods* ativos durante o período de sete dias, como pode ser visto através da Figura 19.

Figura 19 – Média de funcionamento dos Pods nos últimos 7 dias

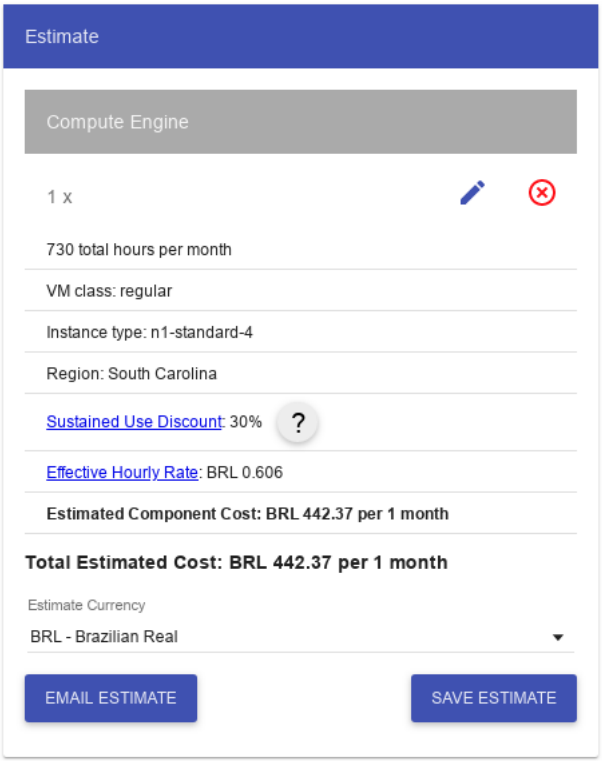


Fonte: Imagem obtida através do monitoramento do Grafana.

[5] O grafana é uma plataforma que permite visualizar dados em tempo real, análisar métricas e monitorá-las por meio dashborads.

Com esses dados, observou-se que a aplicação como um todo mantém, em média, 4 pods ativos simultaneamente, já contando com o Integrator, que fica sempre em execução. Tendo em vista que cada classificador exige apenas 1 CPU e 2 GB de memória RAM, seria necessário apenas 1 *node n1-standard-4* para manter a aplicação funcionando, dado que o *node* definido no *cluster* é de 4 CPUs e 15 GB de memória RAM. Com isso, o gasto real da aplicação Sam com a utilização do Knative pode ser visto na Figura 20.

Figura 20 – Custo médio da aplicação Sam com o Knative



Fonte: Google Cloud Platform Pricing Calculator, Google. Elaborado pelo autor.

Comparando os dois casos, é possível notar que sem utilizar o Knative tem-se 100% dos *pods* ativos (16 *pods*) durante todo o tempo, enquanto utilizando o Knative tem-se apenas 25% dos *pods* ativos (4 *pods*) durante o mesmo período. Isto resulta em uma economia de recursos de 75% ao utilizar o Knative especificamente para a aplicação Sam.

Levando isso para os custos, sem a utilização do Knative, a despesa gerada para manter toda a aplicação ativa durante um mês é de R$ 1769,47. Com a utilização do Knative, esses custos são reduzidos, em média, para R$ 442,37, o que gera uma redução de gastos de R$ 1327,10, representando uma economia de 74,9%.

5 Considerações finais

A maioria das empresas estão se esforçando para reduzir seus custos de computação por meio da virtualização, o que instigou as equipes de desenvolvedores a produzirem aplicações orientadas a microsserviços em *containers*, na qual é possível dimensionar recursos para atender às demandas dos usuários. Entretanto, é difícil manter o desempenho e confiabilidade, enquanto a demanda por determinados serviços escala. Assim, o Google desenvolveu o Kubernetes para gerenciar cargas de trabalho e serviços em *containers*.

Este trabalho apresenta um estudo da implantação do Knative como uma ferramenta alternativa para auxiliar na resolução do problema do alto consumo de recursos e de custos elevados de um *cluster* de Kubernetes. Para isso, foi utilizado o Google Cloud Platform, na qual o Knative foi instalado e configurado em um *cluster*. Além disso, algumas aplicações reais foram adaptadas, de modo a serem implantadas utilizando o Knative. Para validação dos resultados, foram realizados testes comparativos em diferentes cenários do impacto gerado pela aplicação, no que se refere à alocação de recursos (CPU e memória RAM), desempenho e redução de custos no *cluster* de Kubernetes.

Os resultados obtidos sugerem que o Knative é uma tecnologia promissora, possibilitando que aplicações implantadas em um *cluster* sejam gerenciadas de forma inteligente, melhorando a utilização de recursos do *cluster* e reduzindo despesas com recursos de computação de forma significativa. Outro ponto observado é que, na versão do Knative estudada, a ferramenta alcança bons resultados no gerenciamento de aplicações que não são de uso crítico e que não recebem grandes cargas de trabalho simultâneas. Uma vez que a ferramenta ainda caminha para ser um ambiente suficientemente maduro, novos testes são necessários para assegurar que a mesma possa ser implantada sem trazer grandes riscos em produção.

Com isso é possível concluir que a principal contribuição deste trabalho foi demonstrar que a implantação do Knative pode ser utilizada afim de reduzir os elevados custos de um *cluster* de Kubernetes. Tal análise é relevante para o mercado de trabalho e para a comunidade acadêmica, uma vez que as discussões sobre esse tipo de implementação no Brasil ainda são pouco exploradas.

5.1 Trabalhos Futuros

5.1.1 Avaliações de outras aplicações

Foi avaliada também uma outra aplicação que processa postagens coletadas de redes sociais e portais de notícias para fazer o enriquecimento dos dados de um banco de dados. Para cada postagem que deve ser processada é feita uma requisição nessa aplicação, seu processamento, porém, é demorado possuindo várias etapas e demandando um alto poder de processamento.

Dessa forma, haviam muitas requisições simultâneas e cada postagem pode necessitar de um poder de processamento diferente e demorar tempos bem variados de acordo com seu conteúdo, com isso o Knative não conseguiu distribuir bem a carga de trabalho entre os *pods* dessa aplicação, deixando o processamento final mais lento.

É possível que esse ponto de escala seja resolvido utilizando-se como parâmetro para *autoscale* o uso de CPU, uma vez que a métrica de número de requisições simultâneas não é um bom parâmetro para *autoscale* neste cenário porque cada postagem varia muito em tempo e necessidade de CPU. Ou até mesmo utilizando métricas customizadas que passarão a ser suportadas no Knative segundo seu *roadmap* de desenvolvimento (KNATIVE, 2019a).

Dessa forma, seria interessante testar outros tipos de aplicações utilizando diferentes parâmetros para o *autoscale*, com o intuito de avaliar como elas se comportariam utilizando o Knative, no que diz respeito a desempenho e estabilidade.

5.1.2 Melhorias e novas funcionalidades da tecnologia

O Knative apresenta em seu *roadmap* de desenvolvimento funcionalidades e melhorias muito interessantes para o contexto em que o utilizamos neste trabalho (KNATIVE, 2019a). São entregas planejadas com o foco principal em escalabilidade através de melhorias em performance, confiabilidade, extensibilidade e experiência do usuário.

No que diz respeito a performance, tem-se o objetivo de *Sub-Second Cold Start* em que se deseja otimizar a inicialização de *pods* para levar menos de 1 segundo, sendo que atualmente leva-se de 10 a 15 segundos para iniciar de um cenário em que nenhum *pod* da aplicação está em execução. Há também o objetivo de *Overload Handling* que busca um melhor gerenciamento de sobrecarga nos *pods*, atualmente quando um *pod* atinge seu limite de requisições simultâneas, requisições subsequentes são rejeitadas com erro 503 caso não haja *pods* disponíveis ou suficientes para respondê-la, a ideia aqui é implementar uma fila de requisições de tamanho configurável antes das requisições chegarem nos *pods* de modo que novos *pods* escalados ajudariam já a partir do momento que fossem criados consumindo dessa fila e não somente para requisições futuras que ainda irão ocorrer.

Nos pontos de confiabilidade, temos foco em disponibilidade e escalabilidade do

controlador de *autoscale* do próprio Knative. Com foco em extensibilidade, temos o objetivo de permitir a capacidade de criar componentes customizados que estendem o funcionamento do sistema de *autoscale* do Knative. E por fim, tem-se um foco em experiência do usuário ao buscar facilitar a migração de *deployments* de Kubernetes para o Knative através de documentação e exemplos.

Com isso, é possível ver como o desenvolvimento e futuro do Knative é levado a sério e é promissor para todos seus usuários e até mesmo para aqueles que ainda não adotaram esta tecnologia.

5.1.3 Integrações com outras tecnologias

O responsável por entregar as requisições para o Knative é o Istio, que é o componente de *service mesh* utilizado por padrão e recomendado pela documentação.

Como o Istio é o responsável por fazer o gerenciamento do tráfego e a comunicação entre os microsserviços e aplicações, o fato do Knative não conseguir resolver a necessidade de processar uma grande carga de trabalho pode ser um problema na interação com o Istio.

Outra ferramenta recomendada na documentação do Knative é o Gloo. Testes com este ou outros componentes de *service mesh* podem demonstrar melhores resultados.

Uma outra linha de pesquisa é a implementação de um sistema que combine o Knative com outras tecnologias, como por exemplo o Keel, uma ferramenta que automatiza *deploys* no Kubernetes. Com essa combinação, há a possibilidade de atualizar e realizar *deploys* automáticos do Keel junto ao *autoscale* de aplicações e máquinas fornecido pelo Knative deixando o ambiente como um todo muito mais automatizado.

Referências

AMAZON. *Computação em núvem*. 2019. <https://aws.amazon.com/pt/what-is-cloud-computing/> (Accessed: Jun 16, 2019). Citado na página 22.

BRYANT, D. *Google lança Knative: Framework Kubernetes para Build, Deploy e gerenciar serveless workloads*. 2018. <https://www.infoq.com/br/news/2018/09/knative-kubernetes-serverless/> (Accessed: Jun 28, 2019). Citado 4 vezes nas páginas 14, 26, 27 e 28.

BURT, J. *Red Hat, Google, IBM, and SAP go Knative for serveless*. 2018. <https://www.nextplatform.com/2018/12/10/red-hat-google-ibm-and-sap-go-knative-for-serverless/> (Accessed: Jun 27, 2019). Citado na página 26.

GENS, F. *IDC FutureScape: Worldwide IT Industry 2017 Predictions*. [S.l.]: Framingham, UK: International Data Corporation, 2016. Citado na página 23.

GLIKSON, A.; NASTIC, S.; DUSTDAR, S. Deviceless edge computing: extending serverless computing to the edge of the network. In: ACM. *Proceedings of the 10th ACM International Systems and Storage Conference*. [S.l.], 2017. p. 28. Citado na página 25.

GOOGLE. *GKE Overview*. 2019. <https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview> (Accessed: Jun 24, 2019). Citado na página 30.

GOOGLE. *Sustained Use Discounts*. 2019. <https://cloud.google.com/compute/docs/sustained-use-discounts> (Accessed: Jul 08, 2019). Citado na página 44.

GOOGLE. *Sustained Use Discounts*. 2019. <https://cloud.google.com/compute/all-pricing#machinetype> (Accessed: Jul 08, 2019). Citado na página 44.

KAVIANI, N.; KALININ, D.; MAXIMILIEN, M. Towards serverless as commodity: a case of knative. In: ACM. *Proceedings of the 5th International Workshop on Serverless Computing*. [S.l.], 2019. p. 13–18. Citado na página 28.

KNATIVE. *2019 Autoscaling Roadmap*. 2019. <https://github.com/knative/serving/blob/master/docs/roadmap/scaling-2019.md> (Accessed: Dez 02, 2019). Citado na página 50.

KNATIVE. *Configurin the autoscaler*. 2019. <https://knative.dev/docs/serving/configuring-the-autoscaler/> (Accessed: Nov 05, 2019). Citado na página 36.

KNATIVE. *Install on Google Kubernetes Engine*. 2019. <https://knative.dev/docs/install/knative-with-gke/> (Accessed: Jun 25, 2019). Citado 2 vezes nas páginas 32 e 44.

KNATIVE. *Welcome to Knative*. 2019. <https://knative.dev/docs/> (Accessed: Jun 25, 2019). Citado na página 26.

KOLLER, R.; WILLIAMS, D. Will serverless end the dominance of linux in the cloud? In: ACM. *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. [S.l.], 2017. p. 169–173. Citado na página 26.

LYNN, T. et al. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In: IEEE. *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.], 2017. p. 162–169. Citado 2 vezes nas páginas 22 e 25.

MANOR, E. *Bringing the best of serverless to you*. 2018. <https://cloudplatform.googleblog.com/2018/07/bringing-the-best-of-serverless-to-you.html> (Accessed: Jun 28, 2019). Citado na página 27.

MICROSOFT. *O que é computação em nuvem?* 2019. <https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing> (Accessed: Jun 23, 2019). Citado 2 vezes nas páginas 23 e 25.

REDHAT. *Introdução aos containers Linux*. 2019. <https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container> (Accessed: Mai 10, 2019). Citado 3 vezes nas páginas 14, 17 e 18.

REDHAT. *O que é Docker?* 2019. <https://www.redhat.com/pt-br/topics/containers/what-is-docker> (Accessed: Jun 23, 2019). Citado na página 19.

REDHAT. *Vantagens do Kubernetes*. 2019. <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes> (Accessed: Jun 16, 2019). Citado 2 vezes nas páginas 14 e 19.

RIBAS, M. et al. Tomada de decisão multicritério na migração de aplicativos para ambientes de nuvem do tipo software as a service. *Revista Brasileira de Administração Científica*, v. 5, n. 2, p. 83–94, 2014. Citado na página 13.

SAYFAN, G. *Mastering Kubernetes*. [S.l.]: Packt Publishing Ltd, 2017. Citado na página 20.

SCURRA. *Infraestrutura e serviços para cloud: sobre a nuvem híbrida*. 2017. <http://www.scurra.com.br/blog/infraestrutura-e-servicos-para-cloud-sobre-nuvem-hibrida/> (Accessed: Jun 23, 2019). Citado na página 23.

SILVA, M. N. da; CARVALHO, M. Análise de mecanismos de serverless computing em ambientes de nuvens computacionais. In: SBC. *Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2019. p. 225–232. Citado na página 29.

TURNBULL, J. *The Docker Book, Containerization is the new virtualization*. [S.l.]: Copyright 2015, 2017. Citado na página 18.