

INSTITUTO FEDERAL DE SANTA CATARINA

MATUZALEM MÜLLER DOS SANTOS

**Proposta de armazenamento distribuído
baseado em contêineres em infraestrutura
hiperconvergente**

São José - SC

Fevereiro/2019

PROPOSTA DE ARMAZENAMENTO DISTRIBUÍDO BASEADO EM CONTÊINERES EM INFRAESTRUTURA HIPERCONVERGENTE

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Ederson Torresini

Coorientador: Jorge Henrique Busatto Casagrande

São José - SC

Fevereiro/2019

Matuzalem Müller dos Santos

Proposta de armazenamento distribuído baseado em contêineres em infraestrutura hiperconvergente/ Matuzalem Müller dos Santos. – São José - SC, Fevereiro/2019-
95 p. : il. (algumas color.) ; 30 cm.

Orientador: Ederson Torresini

Coorientador: Jorge Henrique Busatto Casagrande

Monografia Engenharia de Telecomunicações – Instituto Federal de Santa Catarina – IFSC

Campus São José

Engenharia de Telecomunicações, Fevereiro/2019.

1. Nuvem Privada. 2. Contêineres. 3. Hiperconvergência de Infraestrutura. I. Ederson Torresini. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Proposta de armazenamento distribuído baseado em contêineres em infraestrutura hiperconvergente

MATUZALEM MÜLLER DOS SANTOS

**PROPOSTA DE ARMAZENAMENTO DISTRIBUÍDO
BASEADO EM CONTÊINERES EM
INFRAESTRUTURA HIPERCONVERGENTE**

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 15 de fevereiro de 2019

Ederson Torresini, Me.
Orientador
Instituto Federal de Santa Catarina

Emerson Ribeiro de Mello, Dr.
Professor
Instituto Federal de Santa Catarina

Humberto José de Sousa
Analista de TI
Instituto Federal de Santa Catarina

*Este trabalho é dedicado a meu pai e minha mãe,
que me ensinaram o valor do trabalho árduo e honesto
e me orgulham todos os dias.*

AGRADECIMENTOS

A Deus por me dar saúde e força para completar o curso.

A meu pai, que sempre me incentivou a buscar o conhecimento.

A minha mãe, que sempre me mostrou que a bondade e gentileza são importantes não apenas no seio familiar, mas também em minha caminhada acadêmica e profissional.

A meu irmão, meu melhor amigo e exemplo de que o estudo e dedicação trazem os resultados merecidos.

A meus amigos e colegas de curso pelas conversas, palavras de apoio, trabalhos em grupo e ajuda nas horas difíceis.

Ao IFSC pela alta qualidade de ensino e surpreendente capacidade de continuar melhorando a qualidade dos cursos ofertados.

A todos os professores do curso, que durante os últimos anos foram grandes fontes de conhecimento e tornaram possível que este trabalho pudesse ter sido realizado.

Aos professores Ederson Torresini e Jorge Henrique Busatto Casagrande, que além de serem meus orientadores também provaram serem grandes amigos, seja com suas palavras de conforto, motivação e entusiasmo, ou compartilhando minhas frustrações, objetivos e metas.

“Um líder sábio é capaz de encontrar algo de bom mesmo nos comentários de um louco.”
(Wei Zheng)

RESUMO

Infraestruturas de serviços baseadas em contêineres têm se popularizado nos últimos anos desde o lançamento da primeira versão estável do Kubernetes pela Google em 2014. Dentre os benefícios oferecidos por tais infraestruturas caracterizam-se a alta escalabilidade e facilidade no gerenciamento de serviços. Este trabalho propõe uma infraestrutura hiperconvergente baseada em contêineres. Para validação da proposta foi criado um *cluster* Kubernetes na *Google Cloud Platform (GCP)*, onde o Rook, uma plataforma de armazenamento distribuído, foi utilizado para validar o gerenciamento de armazenamento na infraestrutura hiperconvergente. Para utilização do armazenamento foram utilizados um blog WordPress e uma base de dados MySQL.

Palavras-chave: Nuvem Privada. Contêineres. Hiperconvergência de Infraestrutura.

ABSTRACT

Container-based infrastructures became popular over the past few years since Google released Kubernetes in 2014. Among the advantages of container-based infrastructures are the immense scaling and easy management capabilities offered by this technology. This paper presents a hyper-converged infrastructure model based on containers. A Kubernetes cluster is implemented in the Google Cloud Platform to validate the model, and Rook (a storage orchestrator) is used to provide storage to applications from the cluster. A WordPress blog and a MySQL database are also implemented to use the storage orchestrated by Rook.

Keywords: Private cloud. Containers. Hyper-converged infrastructure.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelos de serviço em nuvem	33
Figura 2 – Topologia convencional de um <i>data center</i>	36
Figura 3 – Comparação entre máquinas virtuais e contêineres	38
Figura 4 – Arquitetura de <i>cluster</i> Kubernetes	40
Figura 5 – Objetos do Kubernetes	41
Figura 6 – Infraestrutura hiperconvergente de contêineres	46
Figura 7 – Máquinas virtuais criadas na GCP	50
Figura 8 – Saída do terminal: Nós em execução após criação de <i>cluster</i> com Rancher	51
Figura 9 – Representação da arquitetura do Rook	52
Figura 10 – Saída do terminal: <i>Pods</i> criados pelo Rook <i>Operator</i>	53
Figura 11 – Saída do terminal: <i>Pods</i> criados pelo <i>Cluster Rook</i>	54
Figura 12 – Saída do terminal: <i>Pod Reliable Autonomic Distributed Object Store</i> (RADOS) <i>Gateway</i> em execução	55
Figura 13 – Saída parcial do terminal: Requisição de volume e volume utilizado pelo <i>pod MySQL</i>	56
Figura 14 – Saída do terminal: <i>Pod MySQL</i> em execução	56
Figura 15 – Saída parcial do terminal: Requisição de volume e volume utilizado pelo <i>pod WordPress</i>	56
Figura 16 – Saída do terminal: <i>Pod WordPress</i> em execução	57
Figura 17 – <i>Blog</i> WordPress	57
Figura 18 – Imagem armazenada como objeto	58
Figura 19 – Publicação WordPress utilizando imagem armazenada como objeto	59
Figura 20 – Esquemático da implementação	59

LISTA DE TABELAS

Tabela 1 – Especificação das máquinas virtuais	48
Tabela 2 – Especificação da sub-rede	48

LISTA DE CÓDIGOS

Código 3.1 – Configuração de máquina virtual descrita no arquivo de configuração do Terraform	48
Código 3.2 – Configuração de nó mestre do <i>cluster</i> criado com Rancher	50

LISTA DE ABREVIATURAS E SIGLAS

AKS <i>Azure Kubernetes Service</i>	39
Amazon EKS <i>Amazon Elastic Container Service for Kubernetes</i>	39
API <i>Application Program Interface</i>	41
AS <i>Aggregation Switch</i>	37
AWS <i>Amazon Web Services</i>	28
CDC Centros de Dados Compartilhados	28
CI Infraestrutura Convergente	45
CNCF <i>Cloud Native Computing Foundation</i>	39
CR <i>Core Router</i>	37
CRD <i>Custom Resource Definition</i>	41
CRUSH <i>Controlled Replication Under Scalable Hashing</i>	43
CTIC Coordenadoria de Tecnologia da Informação e Comunicação	29
DNS <i>Domain Name System</i>	54
IaaS Infraestrutura como Serviço	33

IFSC Instituto Federal de Santa Catarina.....	29
IP <i>Internet Protocol</i>	34
GCP <i>Google Cloud Platform</i>	11
GFS <i>Google File System</i>	43
GKE <i>Google Kubernetes Engine</i>	39
HCI Infraestrutura Hiperconvergente	45
HTTP <i>Hypertext Transfer Protocol</i>	54
MDS <i>Ceph Metadata Server</i>	43
OSD <i>Object Storage Daemon</i>	43
PaaS Plataforma como Serviço	32
PV <i>Persistent Volume</i>	40
PVC <i>Persistent Volume Claim</i>	40
RADOS <i>Reliable Autonomic Distributed Object Store</i>	15
REST <i>Representational State Transfer</i>	44
RGW <i>RADOS Gateway</i>	44
RKE <i>Rancher Kubernetes Engine</i>	42

RNP <i>Rede Nacional de Ensino e Pesquisa</i>	28
SaaS <i>Software como Serviço</i>	32
SSH <i>Secure Shell</i>	49
TCP <i>Transmission Control Protocol</i>	41
TI <i>Tecnologia da Informação</i>	27
TIC <i>Tecnologia da Informação e Comunicação</i>	28
ToR <i>Top-of-Rack</i>	37
UDP <i>User Datagram Protocol</i>	48
vCPUs <i>Virtual Central Processing Units</i>	48
VLANs <i>Virtual Local Area Networks</i>	29

SUMÁRIO

1	INTRODUÇÃO	27
1.1	Motivação	30
1.2	Justificativa	30
1.3	Objetivos	30
1.3.1	Objetivos Específicos	30
2	FUNDAMENTAÇÃO TEÓRICA	31
2.1	Computação em Nuvem	31
2.1.1	Características Essenciais	31
2.1.2	Modelos de Implantação	32
2.1.3	Modelos de Serviço	32
2.2	Sistemas Distribuídos	33
2.2.1	Virtualização de Sistemas Operacionais	34
2.2.1.1	Tipos de Virtualização de Sistema Operacional	35
2.2.2	Virtualização de Redes	36
2.2.3	Contêineres	37
2.2.3.1	Orquestração de Contêineres	39
2.2.3.1.1	Kubernetes	39
2.2.3.1.2	Helm	41
2.2.3.2	Rancher	42
2.3	Sistemas de Arquivos	42
2.3.1	Sistemas de Arquivos Distribuídos	42
2.3.1.1	Ceph	43
2.3.1.2	Rook	44
2.3.2	Convergência de infraestrutura	45
3	IMPLEMENTAÇÃO DO SISTEMA DE ARMAZENAMENTO DISTRIBUÍDO	47
3.1	Criação da infraestrutura remota	47
3.2	Criação do <i>cluster</i> Kubernetes	50
3.3	Implantação do Rook para utilização de armazenamento	52
3.4	Implantação de serviços para utilizar armazenamento	55
3.4.1	MySQL	55
3.4.2	WordPress	55
4	CONSIDERAÇÕES FINAIS	61

4.1	Trabalhos futuros	61
	REFERÊNCIAS	63
	APÊNDICES	69
	APÊNDICE A – <i>SCRIPT DE INICIALIZAÇÃO DAS MÁQUINAS VIRTUAIS</i>	71
	APÊNDICE B – ARQUIVO DE CONFIGURAÇÃO DO TERRAFORM	73
	APÊNDICE C – ARQUIVO DE CONFIGURAÇÃO DO RANCHER	77
	APÊNDICE D – ARQUIVO DE CRIAÇÃO DO <i>CLUSTER ROOK</i> .	79
	APÊNDICE E – ARQUIVO DE INICIALIZAÇÃO DO <i>STORAGE-CLASS ROOK</i>	81
	APÊNDICE F – ARQUIVO <i>INGRESS DA OBJECT STORE</i>	83
	ANEXOS	85
	ANEXO A – INSTRUÇÕES DE CRIAÇÃO DO <i>CLUSTER KUBERNETES</i>	87
	ANEXO B – INSTRUÇÕES DE IMPLANTAÇÃO DO <i>CLUSTER ROOK</i>	91

1 INTRODUÇÃO

Segundo Mell e Grance (2011), computação em nuvem é um modelo que possibilita acesso através da rede a recursos (como armazenamento, por exemplo) de forma universal, conveniente e sob demanda. Estes recursos podem ser rapidamente alocados e descartados com baixo esforço e interação com o provedor do serviço.

No decorrer da última década notou-se um grande aumento na utilização de serviços em nuvem. Segundo Kavis (2014), em 2013 a maior parte das *startups* e empresas de médio porte estavam desenvolvendo suas aplicações para prover serviços em nuvem com suas infraestruturas localizadas em nuvens públicas. A utilização de nuvens públicas têm se tornado cada vez mais comum por conta dos diversos benefícios ofertados. Dentre estes benefícios, tem-se:

- O modelo de utilização *pay-as-you-go*, onde a empresa paga apenas pelos recursos que utiliza da nuvem, ao invés de comprar inúmeros equipamentos e ter que arcar com a manutenção destes equipamentos mesmo quando não utilizados ou ociosos;
- A facilidade de implantação de serviços, haja vista a abstração da infraestrutura pelo provedor;
- A contratação de um serviço especializado em *data centers* e infraestruturas oferecerá uma infraestrutura muito mais confiável e tolerante a falhas do que a de uma nuvem privada da empresa em questão, principalmente se o fim da empresa não for Tecnologia da Informação (TI) e se ela já não possuir uma infraestrutura de alta disponibilidade.

Esta última vantagem também permite que desenvolvedores e funcionários de TI se concentrem no serviço ofertado pela empresa, e não em manter a infraestrutura (em nível de *hardware*) do sistema. Os funcionários de TI, por exemplo, poderiam assumir o papel de administradores do sistema, enquanto os desenvolvedores poderiam concentrar seu tempo no desenvolvimento de novas tecnologias.

Por estes motivos, grandes empresas também estão migrando suas infraestruturas para nuvens públicas com o intuito de melhorar a eficiência de seus serviços e diminuir gastos de manutenção e expansão. Também de acordo com Kavis (2014), um exemplo de grande empresa que realizou a migração de seus serviços de uma nuvem privada para uma nuvem pública é a Netflix. Em 2009, 100% do tráfego de usuários circulava dentro do próprio *data center* da empresa. Com o aumento do número de usuários do serviço de *streaming*, a empresa precisou reestruturar sua arquitetura interna para adequar-se

à elevada carga de tráfego gerada pelo *streaming* de mídia. Por este motivo, em 2008 a empresa iniciou a migração de sua infraestrutura para a *Amazon Web Services* ([AWS](#)). Segundo [Izrailevsky \(2016\)](#), em janeiro de 2016 esta migração foi finalizada, e atualmente toda a infraestrutura da Netflix está localizada na nuvem pública da *Amazon*.

Apesar dos diversos benefícios ofertados por nuvens públicas, estas nem sempre são a melhor solução para armazenamento e processamento de dados. Órgãos governamentais possuem grandes quantidades de dados, mas ainda há barreiras na utilização de nuvens públicas por estas entidades. Com foco neste debate, foi publicado o “Manual de Boas práticas, orientações e vedações para contratação de Serviços de Computação em Nuvem” pelo [Departamento de Infraestrutura e Serviços de Tecnologia da Informação \(2016\)](#). Este manual veda a utilização de nuvens públicas em alguns cenários, como em casos em que os serviços de Tecnologia da Informação e Comunicação ([TIC](#)) possam comprometer a segurança nacional, por exemplo. Uma das preocupações na contratação de serviços em nuvem por órgãos governamentais expressas no manual consiste no controle sobre os dados armazenados por terceiros, os quais devem:

“[...] residir exclusivamente em território nacional, incluindo replicação e cópias de segurança (backups), de modo que o contratante disponha de todas as garantias da legislação brasileira enquanto tomador do serviço e responsável pela guarda das informações armazenadas em nuvem.” ([Departamento de Infraestrutura e Serviços de Tecnologia da Informação, 2016](#))

Portanto, é aconselhável que duas condições sejam satisfeitas na contratação de serviços em nuvens públicas por órgãos governamentais: a primeira é de que o serviço contratado atenda a todos os requisitos de geolocalização, privacidade, segurança e qualidade de serviço determinados pelo contratante; a segunda condição refere-se à qualidade e continuidade na gestão sobre o serviço contratado, ou seja, é necessário que seja realizado um planejamento a longo prazo de como o serviço em nuvem será integrado e mantido com o sistema atual. Este planejamento deve ser realizado e seguido de acordo com as necessidades dos órgãos, e não ser alterado por mudanças de gestão ou pessoal, o que dificultaria a implantação e integração do serviço em nuvem com o sistema. Já em relação ao fornecimento de serviços em nuvem por terceiros, algumas empresas têm buscado certificações e oferecido serviços específicos para adequar-se a demandas específicas. Um bom exemplo de como provedores de serviços em nuvem estão começando a prover serviços específicos para governos é a *Amazon GovCloud*, a qual foi projetada para oferecer serviços em nuvem para o governo estadunidense ([DIEZ; SILVA, 2013](#)). No Brasil, a Rede Nacional de Ensino e Pesquisa ([RNP](#)) está trabalhando nos Centros de Dados Compartilhados ([CDC](#)) para fornecimento de serviços de computação em nuvem para instituições de ensino e pesquisa no Brasil ([Rede Nacional de Pesquisa, 2014](#)). Entretanto, este projeto ainda não atende

a todas as instituições de ensino e pesquisa do país, o que torna a utilização de nuvens privadas comum dentro destas instituições e de outros órgãos governamentais.

Além de poder oferecer um maior nível de controle sobre os dados armazenados, nuvens privadas também possuem outras vantagens. Como os administradores dos *data centers* possuem total controle sobre os serviços executados e dados armazenados, é possível rodar serviços específicos em servidores específicos, que possuam o *hardware* dedicado ou que melhor se encaixem no perfil do serviço utilizado ou ofertado. Além disso, nuvens públicas geralmente realizam o gerenciamento de arquivos através de objetos, sendo serviços transparentes ao usuário (AWS, 2018) (Microsoft Azure, 2018). Já em nuvens privadas, o gerenciamento de arquivos é construído pelos próprios administradores da nuvem, podendo oferecer diversos tipos de armazenamento (em objetos, arquivos ou blocos). Este controle de desempenho no acesso a dados em nuvens privadas também favorece a utilização de nuvens híbridas, onde parte da infraestrutura do sistema está localizada em uma nuvem privada e outra parte localizada em uma nuvem pública. O armazenamento de dados pode, portanto, ser realizado em uma rede privada otimizada, e servido a aplicações que operem em uma nuvem pública, por exemplo.

Em todos os cenários exemplificados acima, a virtualização pode ser utilizada para melhorar a eficiência do sistema. Seja com a utilização de máquinas virtuais para aumentar o grau de portabilidade, flexibilidade e segurança do sistema (TANENBAUM; STEEN, 1994) ou na implementação de técnicas de tunelamento e *Virtual Local Area Networks* (VLANs) para melhor gerenciamento da rede (GÖRANSSON; BLACK, 2014), o conceito de virtualização está presente na maior parte dos *data centers*. Há também uma crescente utilização de contêineres como substitutos a máquinas virtuais, devido a sua baixa sobrecarga na implantação de serviços (MEDEL et al., 2016).

Visando a melhoria do sistema atual e evolução de serviços ofertados à comunidade acadêmica, a Coordenadoria de Tecnologia da Informação e Comunicação (CTIC) do câmpus São José do Instituto Federal de Santa Catarina (IFSC) tem estudado técnicas e implementações de computação em nuvem privada dentro do câmpus (CTIC - IFSC câmpus São José, 2018). Este trabalho apresenta uma infraestrutura hiperconvergente implantada em uma nuvem pública. As tecnologias utilizadas neste trabalho são as mesmas atualmente estudadas pela CTIC do câmpus São José, e espera-se que este trabalho possa contribuir com os estudos de armazenamento distribuído em nuvens de contêineres. A seguir serão apresentadas a motivação, justificativa e objetivos do trabalho, seguidos pela fundamentação teórica e implantação de um sistema de armazenamento distribuído em uma nuvem de contêineres. Ao final do trabalho são realizadas as considerações finais.

1.1 Motivação

Atualmente nuvens privadas e tecnologias de virtualização estão sendo estudadas no câmpus São José do [IFSC](#) para melhorar a eficácia dos serviços ofertados à comunidade acadêmica. O tema de armazenamento em nuvem, principalmente, têm sido estudado para garantir que os dados da instituição sejam armazenados de forma segura e eficiente.

1.2 Justificativa

Políticas governamentais impõem que certas informações sejam armazenadas na nuvem privada da instituição ([Departamento de Infraestrutura e Serviços de Tecnologia da Informação, 2016](#)). Este trabalho apresenta a implantação de armazenamento distribuído em uma infraestrutura hiperconvergente, o qual pode servir de subsídio para tomada de decisão em relação ao armazenamento distribuído do câmpus São José.

1.3 Objetivos

Este trabalho têm por objetivo propor uma solução de armazenamento distribuído e apresentar esta proposta através da implementação de um sistema de armazenamento distribuído em uma nuvem de testes distribuída na [GCP](#).

1.3.1 Objetivos Específicos

Estes são os objetivos específicos do trabalho:

- Estudar a utilização de contêineres para fornecimento de serviços.
- Estudar tecnologias de armazenamento distribuído utilizando contêineres.
- Apresentar uma solução de armazenamento distribuído implantada em uma infraestrutura hiperconvergente.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Computação em Nuvem

Segundo [Mell e Grance \(2011\)](#), a computação em nuvem é definida como:

“[...] um modelo que possibilita acesso através da rede a recursos (por exemplo, redes, servidores, armazenamento, aplicações e serviços) de forma universal, conveniente e sob demanda, que podem ser rapidamente fornecidos e escaláveis com baixo esforço e interação com o provedor do serviço. De acordo com as necessidades e limitações do sistema, nuvens podem ser implantadas de forma privada, pública, híbrida e comunitária. Este modelo de computação é composto de cinco características essenciais, quatro modelos de implantação e três modelos de serviço.”

As características, modelos de implantação e modelos de serviços estão descritos a seguir.

2.1.1 Características Essenciais

- Autoatendimento sob demanda: um usuário da nuvem pode utilizar recursos computacionais sob demanda sem necessidade de intervenção do provedor dos serviços ([BELBERGUI; ELKAMOUN; HILAL, 2017](#));
- Amplo acesso a rede: recursos estão disponíveis através da rede e podem ser acessados através de mecanismos padronizados por plataformas heterogêneas (dispositivos móveis, *tablets* e *laptops*, por exemplo) ([MELL; GRANCE, 2011](#));
- *Pooling* de recursos: os recursos computacionais da nuvem são localizados conjuntamente (seja geograficamente ou virtualmente) em um esforço para servir múltiplos usuários, sem que os usuários tenham conhecimento sobre onde os dados estão de fato armazenados ([DILLON; WU; CHANG, 2010](#));
- Elasticidade rápida: recursos podem ser rapidamente alocados e liberados, sendo que para usuários a capacidade de alocação de recursos geralmente aparenta ser ilimitada a qualquer momento ([MELL; GRANCE, 2011](#));
- Serviços mensuráveis: sistemas em nuvem automaticamente controlam e otimizam recursos ao mensurar e abstrair recursos utilizados por serviços (armazenamento, banda, número de usuários ativos, entre outros). A utilização de recursos pode ser monitorada, controlada e informada, provendo transparência tanto para provedor do serviço quanto para o usuário ([MELL; GRANCE, 2011](#)).

2.1.2 Modelos de Implantação

Nuvens de computação podem ser implementadas de quatro formas distintas, seguindo o modelo de nuvens públicas, privadas, híbridas ou comunitárias.

Em uma nuvem pública os serviços são acessíveis através da *Internet* e disponibilizados por um provedor de serviço, responsável por manter e gerenciar a infraestrutura da nuvem e manter o serviço (BELBERGUI; ELKAMOUN; HILAL, 2017). Estes serviços podem incluir uma variedade de aplicações ou serviços de armazenamento, por exemplo. Exemplos de serviços de processamento fornecidos em nuvens públicas são a *Microsoft Azure Services Platform*, a *Amazon Elastic Compute Cloud* e a GCP (GÖRANSSON; BLACK, 2014).

Ao contrário de nuvens públicas, as nuvens privadas são reservadas para uso exclusivo de uma única organização (BELBERGUI; ELKAMOUN; HILAL, 2017). Esta nuvem pode ser mantida, gerenciada e operada pela organização, terceiros, ou uma combinação de ambos, e pode existir dentro ou fora das premissas da organização (MELL; GRANCE, 2011).

Em nuvens híbridas a infraestrutura da nuvem é composta por duas ou mais infraestruturas de nuvens distintas (privadas, comunitárias ou públicas) que continuam sendo entidades separadas, mas são associadas por padronizações ou tecnologias proprietárias que permitem portabilidade de dados ou aplicações (MELL; GRANCE, 2011).

No modelo de nuvens comunitárias a infraestrutura da nuvem é fornecida para uso exclusivo de uma comunidade específica de usuários de organizações que compartilhem valores, políticas, missões ou requisitos de segurança. Ela pode ser mantida, gerenciada e operada por uma ou mais organizações da comunidade, terceiros, ou uma combinação de ambos, e pode existir dentro ou fora da organização (MELL; GRANCE, 2011).

2.1.3 Modelos de Serviço

Três são os modelos de serviços de nuvens, que procuram atender a necessidades distintas. Estes modelos são exibidos na Figura 1 e explicados a seguir.

O modelo de *Software como Serviço* (*SaaS*) é uma forma de computação em nuvem onde o usuário acessa e utiliza aplicações remotas. Como exibido na Figura 1, o usuário geralmente não possui acesso ao sistema operacional do servidor e não é responsável por manter o serviço, utilizando assim apenas as funcionalidades das aplicações (ARINZE; ANANDARAJAN, 2013).

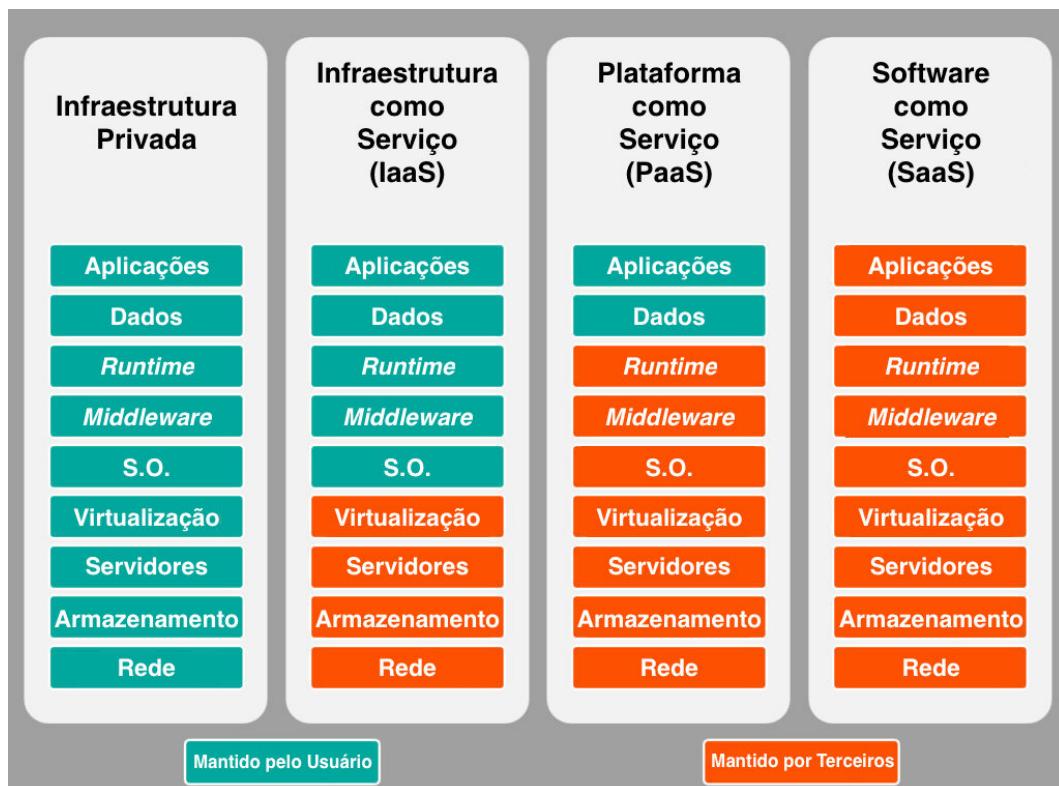
Já no modelo de serviço Plataforma como Serviço (*PaaS*) o provedor do serviço fornece a usuários uma plataforma de computação (geralmente virtual) que provê sistemas operacionais prontos para execução (ARINZE; ANANDARAJAN, 2013). O usuário não

controla ou gerencia a infraestrutura física da nuvem (como rede, servidores ou armazenamento), mas possui controle sobre as aplicações instaladas e configurações destas aplicações (MELL; GRANCE, 2011).

O terceiro modelo, amplamente utilizado no desenvolvimento de aplicações e serviços em nuvem, é o de Infraestrutura como Serviço (IaaS). No IaaS o usuário não controla ou gerencia a infraestrutura física, mas possui controle sobre sistemas operacionais, armazenamento e aplicações, e **possivelmente** controle limitado sobre componentes de rede (como *firewalls*, por exemplo) (MELL; GRANCE, 2011). Sistemas operacionais e aplicações são passíveis de administração pelo usuário, e é de sua responsabilidade a manutenção desses sistemas (ARINZE; ANANDARAJAN, 2013).

Já no modelo de infraestrutura privada o administrador do sistema é responsável por manter todas as camadas, desde a parte física , como servidores e armazenamento, quanto de *software*, como sistema operacional e aplicações.

Figura 1 – Modelos de serviço em nuvem



Fonte: Traduzido de (WATTS, 2017)

2.2 Sistemas Distribuídos

De acordo com Tanenbaum e Steen (1994), um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente. Entre as vantagens na utilização de sistemas distribuídos destacam-se a

escalabilidade e redundância do sistema, que permitem que o sistema atenda à diversas aplicações e usuários. A contrapartida é a complexidade em seu gerenciamento, haja vista que o gerenciamento de recursos e dimensão do sistema ficam mais complexos.

Para que o sistema se apresente ao usuário como um sistema distribuído é necessário que ele cumpra quatro metas fundamentais ([TANENBAUM; STEEN, 1994](#)):

- Acesso a recursos: o acesso a recursos do sistema deve ser eficiente e seguro.
- Transparência da distribuição: o sistema deve se apresentar ao usuário sendo como um único sistema, mesmo estando distribuído entre diversos computadores (por exemplo, ao acessar uma página *Web* o usuário não possui conhecimento sobre qual servidor está de fato acessando).
- Abertura: “um sistema distribuído aberto é um sistema que oferece serviços de acordo com regras padronizadas que descrevem a sintaxe e semântica desses serviços” ([TANENBAUM; STEEN, 1994](#)).
- Escalabilidade: um sistema distribuído pode ser escalável em relação a seu tamanho (com adição de usuários e recursos ao sistema), termos geográficos (usuários e recursos podem estar longes uns dos outros) e administrativos (facilidade de gerenciamento) ([TANENBAUM; STEEN, 1994](#)).

Sistemas de computação distribuídos são classificados em dois subgrupos, sendo estes subgrupos os sistemas de computação em *cluster* e em *grid* ([TANENBAUM; STEEN, 1994](#)). Um *cluster* consiste em um conjunto de computadores de *hardware* semelhante interconectados entre si por uma rede de alta velocidade e que trabalham conjuntamente para executar tarefas que requerem alto poder computacional ([SADASHIV; KUMAR, 2011](#)). Já na computação em *grid* o *hardware* das estações é heterogêneo e as estações que compõem o sistema podem ser de diferentes organizações ([TANENBAUM; STEEN, 1994](#)). Em sistemas de computação em *grid* a estratégia empregada para coordenação entre as estações é a utilização de um *middleware* que divide e distribui tarefas entre os computadores ([SADASHIV; KUMAR, 2011](#)).

Como é possível perceber, é necessário que os computadores que fazem parte do sistema distribuído estejam altamente coordenados para realizar a computação de forma eficiente. Diversos modelos de coordenação são utilizados, como o Linda, o *publish/subscribe* e o Jini ([TANENBAUM, 2010](#)).

2.2.1 Virtualização de Sistemas Operacionais

Com o crescimento de volumes de dados e de serviços fornecidos através de redes *Internet Protocol* ([IP](#)) têm se evidenciado que a virtualização de infraestruturas provê

melhor flexibilidade, custo, escalabilidade, utilização de recursos e eficiência energética em *data centers* (BARI et al., 2013). Até recentemente, a virtualização neste contexto significava apenas virtualização em *hardware* através da execução de máquinas virtuais (TAY; GAURAV; KARKUN, 2017), apesar do conceito de virtualização ser discutido e pesquisado há mais de 40 anos (PEARCE; ZEADALLY; HUNT, 2013). Segundo Walters et al. (2008), a virtualização de sistemas operacionais pode ser majoritariamente categorizada em quatro grupos: virtualização total ou completa, paravirtualização, virtualização em nível de sistema operacional e virtualização assistida por *hardware*. Estes tipos de virtualização são explicados a seguir.

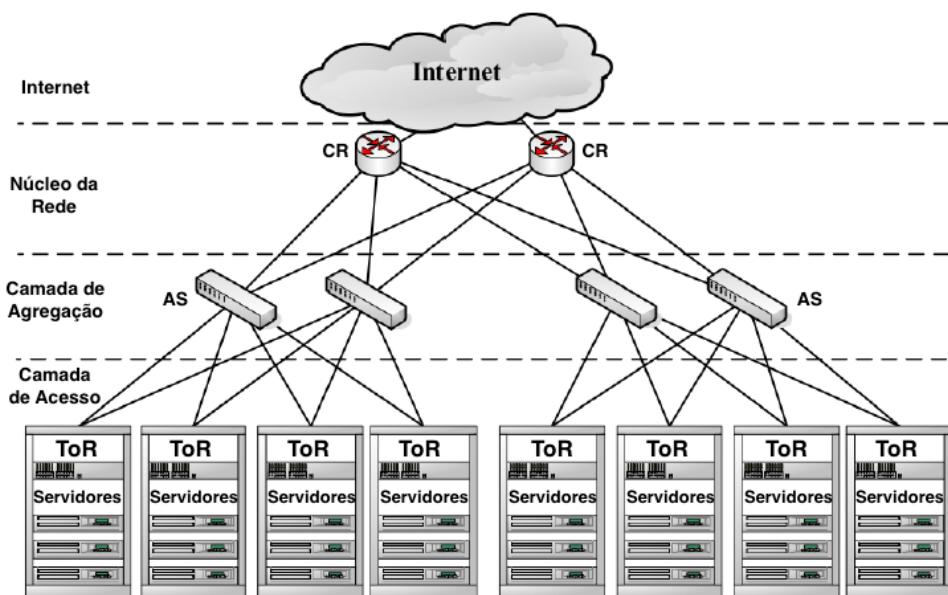
2.2.1.1 Tipos de Virtualização de Sistema Operacional

- Virtualização total ou completa (*Full Virtualization*): Também chamada de emulação de *hardware*. Nesta categoria, o *hypervisor* (*software*, *firmware* ou *hardware* que encapsula e roda o sistema operacional emulado) provê as mesmas entradas, saídas e comportamentos que seriam esperados de um *hardware* específico (PEARCE; ZEADALLY; HUNT, 2013). De forma simples, o sistema operacional emulado não possui conhecimento de que está sendo virtualizado, e sim de que está sendo executado diretamente em *hardware*. Este tipo de virtualização é computacionalmente intensivo e possui alto custo, pois o sistema operacional emulado não possui conhecimento de que está na verdade acessando um *hardware* emulado pelo *hypervisor*, o qual captura as exceções de acesso ao *hardware* do sistema emulado e as emula de acordo com as configurações do *hardware* emulado;
- Paravirtualização (*Paravirtualization*): Na paravirtualização propõe-se que o sistema operacional virtualizado saiba que está sendo executado na camada virtual e interaja com a mesma. Isso implica a alteração do sistema operacional virtualizado, mas garante uma grande cooperação entre as duas camadas (GROSMANN et al., 2011);
- Virtualização em nível de Sistema Operacional (*Operating System-level Virtualization*): De forma simples, esta técnica de virtualização permite que recursos possam ser isolados dentro de um sistema operacional para execução de instâncias de espaços de usuários e aplicações, sendo o gerenciamento destes recursos realizados pelo sistema operacional, e não por um *hypervisor*, o que aumenta o desempenho dessas instâncias (WALTERS et al., 2008);
- Virtualização Assistida por *Hardware* (*Hardware-Assisted Virtualization*): Esta técnica de virtualização consiste em uma virtualização total mas com auxílio em *hardware* do processador, permitindo que as instruções da máquina virtual não sejam executadas em um processador emulado pelo *hypervisor*, e sim diretamente no processador da máquina real (WALTERS et al., 2008).

2.2.2 Virtualização de Redes

A virtualização de servidores sozinha não é suficiente para resolver os problemas e limitações de grandes infraestruturas, tais como os riscos de segurança ocasionados pela não-restrição de protocolos de comunicação utilizados por máquinas reais, as quais podem estar executando diversas máquinas virtuais simultaneamente (BARI et al., 2013). Neste contexto, técnicas de virtualização de rede também têm sido aplicadas com o objetivo de solucionar alguns destes problemas e melhorar a performance e segurança do sistema. Técnicas como a utilização de **VLANs** são as mais tradicionais em termos de virtualização, pois permitem que sejam criadas redes locais virtuais, restringindo assim o domínio de *broadcast* de cada dispositivo e possibilitando a resolução de diversos dos problemas de *data centers* (BARI et al., 2013). Com a utilização de técnicas de virtualização de rede torna-se fundamental a organização de uma topologia que facilite a implementação e manutenção da rede. Ao definir a topologia de uma rede, é importante definir se ela será escalável horizontalmente ou verticalmente. A escalabilidade horizontal refere-se a adicionar nós no sistema, o que permite a distribuição de carga de trabalho a múltiplas máquinas e a utilização de *hardware* heterogêneo no sistema (GREGOL, 2011). Já a escalabilidade vertical refere-se a melhorar o sistema ao adicionar mais recursos a um nó, tais como aumentar a memória ou capacidade de armazenamento do mesmo (BARZU; CARABAS; TAPUS, 2017). A Figura 2 exibe a topologia convencional de *data center* segundo Barzu, Carabas e Tapus (2017).

Figura 2 – Topologia convencional de um *data center*



Fonte: Traduzido de (BARI et al., 2013)

Como exibido na Figura 2, esta topologia é convencionalmente horizontal, pois permite que diferentes dispositivos de uma mesma categoria (*switches*, roteadores e servi-

dores) sejam adicionados ao sistema, evitando assim a necessidade de utilizar tecnologias proprietárias e altos custos na melhoria do sistema, por exemplo. É importante ressaltar que nem sempre esta solução é a ideal, pois ocorrem casos em que melhorar um nó individual é mais efetivo do que adicionar nós ao sistema. Por exemplo, pode ser mais efetivo melhorar a capacidade de processamento de um nó para fornecimento de um serviço específico do que adicionar nós ao sistema. Na topologia apresentada na Figura 2, o *switch Top-of-Rack (ToR)* realiza a conexão entre os servidores montados em um mesmo *rack*, enquanto os *switches* de agregação (*Aggregation Switch (AS)*) encaminham o tráfego dos *switches ToR* para os roteadores do núcleo da rede (*Core Router (CR)*) (BARI et al., 2013). É importante ressaltar a redundância entre os *switches ToR* e de agregação, que garantem que o sistema continuará em operação em caso de falhas pontuais. Esta topologia é denominada topologia *ToR* (referenciando o *switch ToR*, componente vital do sistema).

2.2.3 Contêineres

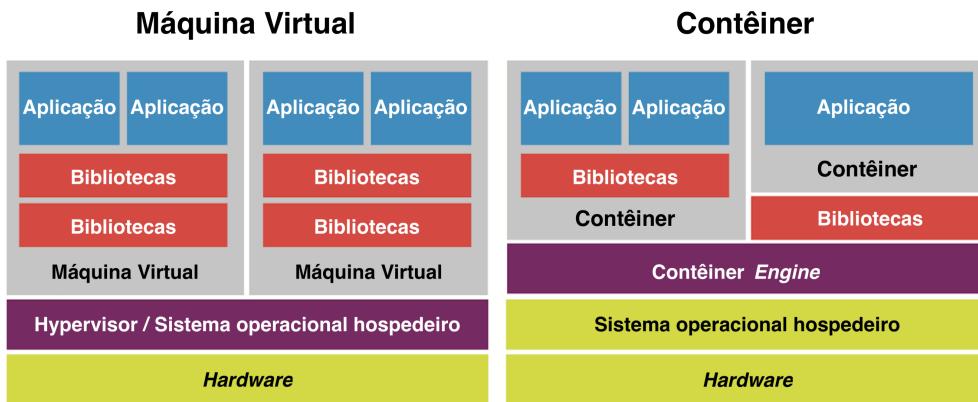
Os avanços no *kernel* do Linux levaram ao desenvolvimento de isolamento de recursos baseado em contêineres (JOY, 2015). De forma simples, um contêiner pode ser definido como um pacote executável de um *software* que inclui tudo o que é necessário para executar este *software*: código, bibliotecas, configurações, ferramentas, entre outros (Docker Inc, 2018b).

Contêineres e máquinas virtuais são soluções distintas para problemas distintos. De acordo com Pahl (2015):

“[...] contêineres são ferramentas para prover software - isto é, seu foco é na entrega de uma *PaaS* - com grande portabilidade e interoperabilidade enquanto utilizando técnicas de virtualização. Por outro lado, máquinas virtuais se preocupam com alocação e gerenciamento de hardware - isto é, seu foco é na entrega de uma *IaaS* que fornece serviços através da virtualização de hardware.”

Existem diferentes tecnologias de contêineres, sendo o Docker uma das tecnologias que está se tornando uma das mais utilizadas para criação de contêineres (AHN et al., 2018). O Docker é uma ferramenta que facilita a criação e distribuição de contêineres com suas dependências (SINGH; SINGH, 2016), sendo um projeto de código aberto que possui mais de 3 mil colaboradores (Docker Inc, 2018a). A Figura 3 exibe a abstração das camadas de máquinas virtuais e contêineres Docker. É possível perceber que, diferentemente de máquinas virtuais, contêineres não requisitam que seja realizada a virtualização completa de um sistema operacional. A vantagem desta abordagem é de que não há duplicação de funcionalidades, como por exemplo chamadas de *hardware*, já que apenas um sistema operacional lida com todo o acesso ao *hardware* (SINGH; SINGH, 2016).

Figura 3 – Comparaçāo entre máquinas virtuais e contēineres



Fonte: Traduzido de ([PAHL, 2015](#))

Como contēineres - diferentemente de máquinas virtuais - rodam como processos em um mesmo sistema operacional, é possível (e necessário) restringir e limitar o acesso a recursos do sistema operacional hospedeiro. Para contēineres em Linux, o controle ao acesso de recursos é realizado através de *namespaces* e grupos de controle (*cgroups*) ([SINGH; SINGH, 2016](#)). *Namespaces* permitem que grupos de processos sejam separados, evitando assim que estes processos acessem recursos de outros grupos ([PAHL, 2015](#)). Grupos de controle permitem o gerenciamento de recursos por grupos de processos, possibilitando que o uso de memória seja limitado para um grupo, por exemplo ([PAHL, 2015](#)).

Segundo [Joy \(2015\)](#), quatro são as características que tornam contēineres úteis e atrativos no desenvolvimento de aplicações e fornecimento de serviços:

- Portabilidade: aplicações podem ser desenvolvidas em um ambiente e ser implantadas em diversos outros ambientes sem necessidade de alterar o contêiner.
- Rapidez na entrega da aplicação: o fluxo de desenvolvimento de contēineres permite que administradores de sistema, desenvolvedores e equipe de testes efetuem a implantação da aplicação rapidamente, pois apenas os desenvolvedores precisam se preocupar com o formato e conteúdo do contêiner e os administradores de sistema com sua implantação nos servidores.
- Escalabilidade: a escalabilidade vertical e horizontal de contēineres é muito mais rápida do que a de máquinas virtuais, já que os recursos utilizados pelos contēineres são oferecidos nativamente pelo sistema operacional, sem necessidade de virtualização de chamadas de *hardware*.
- Maior carga de trabalho e densidade: como não é necessário utilizar um *hypervisor* para virtualização, é possível executar uma quantidade maior de contēineres do que de máquinas virtuais concorrentemente. Além disso, o processamento está concentrado

na aplicação, sem necessidade de virtualização de chamadas de *hardware*, como no caso de máquinas virtuais, por exemplo.

Após a criação do contêiner, é possível gerenciá-lo utilizando uma ferramenta de orquestração de contêineres, que permite a criação de *clusters* de contêineres e seu gerenciamento.

2.2.3.1 Orquestração de Contêineres

De acordo com Khan (2017), uma plataforma de orquestração de contêineres pode ser, de forma geral, definida como um sistema que provê um *framework* para integração e gestão de contêineres em larga escala. Estas plataformas simplificam a gestão de contêineres e fornecem um *framework* para não apenas realizar a implantação inicial de contêineres mas também a gestão de múltiplos contêineres como uma entidade, seja para propósitos de escalabilidade ou disponibilidade, por exemplo. Como exemplos de orquestradores de contêineres populares pode-se citar o Docker Swarm e o Kubernetes, enquanto *Google Kubernetes Engine* (GKE), *Amazon Elastic Container Service for Kubernetes* (Amazon EKS) e *Azure Kubernetes Service* (AKS) são implementações do orquestrador Kubernetes em nuvens públicas.

2.2.3.1.1 Kubernetes

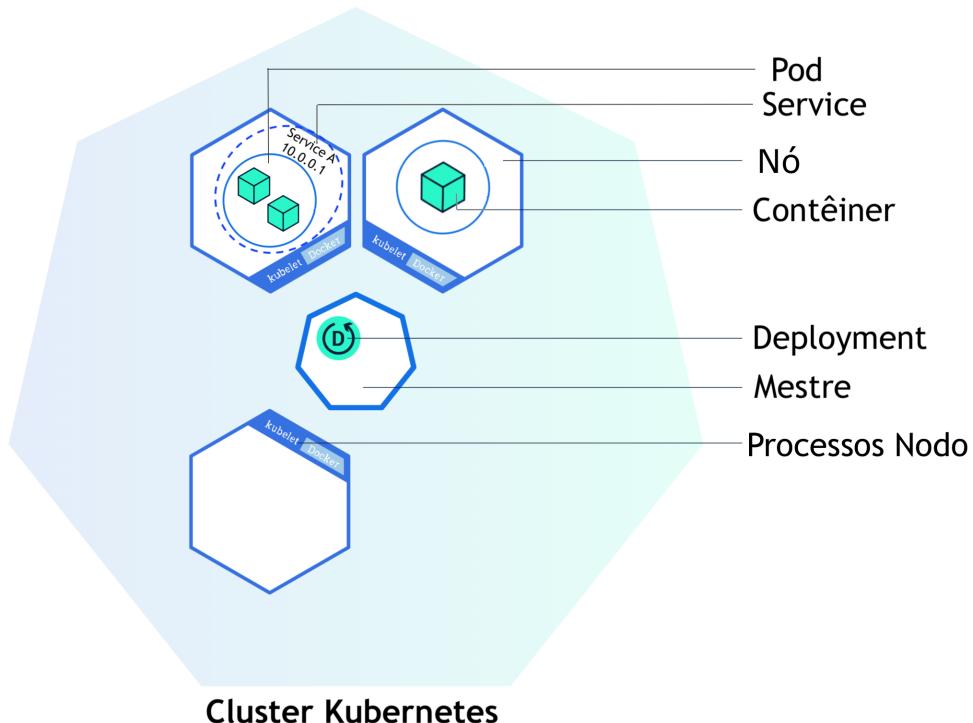
Kubernetes é uma plataforma de código aberto para orquestração de contêineres a qual facilita tanto configurações declarativas e automatizações. Disponibilizada em 2014 e criada pela Google, hoje a plataforma é mantida pela *Cloud Native Computing Foundation* (CNCF) (The Kubernetes Authors, 2018i).

Um *cluster* Kubernetes consiste em dois tipos de recursos: mestres e nós. A interligação entre estes recursos coordenados por um mesmo orquestrador Kubernetes é denominado *cluster* (The Kubernetes Authors, 2018h). Os mestres são responsáveis por coordenar o *cluster*, enquanto os nós são as unidades que de fato executam as aplicações do *cluster*, as quais são implantadas através de contêineres. A Figura 4 mostra a arquitetura básica de um *cluster* Kubernetes.

Na arquitetura da Figura 4 há um mestre e três nós. Os *pods*, *deployments* e *services* são entidades persistentes do sistema Kubernetes e são tratados como objetos (The Kubernetes Authors, 2018f). Já o processo *kubelet* é o “agente” que roda no nó para que este faça parte do *cluster* (The Kubernetes Authors, 2018b).

Um *pod* é um grupo de um ou mais contêineres que compartilham armazenamento, rede e outros recursos computacionais e especifica como executar estes contêineres. Assim como contêineres, *pods* são isolados através da utilização de *namespaces* linux, *cgroups*

Figura 4 – Arquitetura de *cluster* Kubernetes



Fonte: Adaptado de ([The Kubernetes Authors, 2018h](#))

e outras formas de isolamento. *Pods* podem ser utilizados para implantar *proxies* e ferramentas de monitoramento, por exemplo ([The Kubernetes Authors, 2018c](#)).

Services são uma abstração que definem as políticas de acesso a *pods*. *Pods* podem ser expostos através de diferentes tipos de *services*, havendo desde *services* que expõem uma porta do nó externamente ao *cluster* (*NodePort*) até *services* que limitam o acesso ao *pod* a endereços IP internos ao *cluster* (*clusterIP*) ([The Kubernetes Authors, 2018d](#)).

Já um *deployment* permite que *pods* sejam criados e atualizados através de linguagem declarativa. *Deployments* fornecem um mecanismo de auto-reparação que permite que *pods* sejam reiniciados caso falhas ocorram, por exemplo ([The Kubernetes Authors, 2018g](#)).

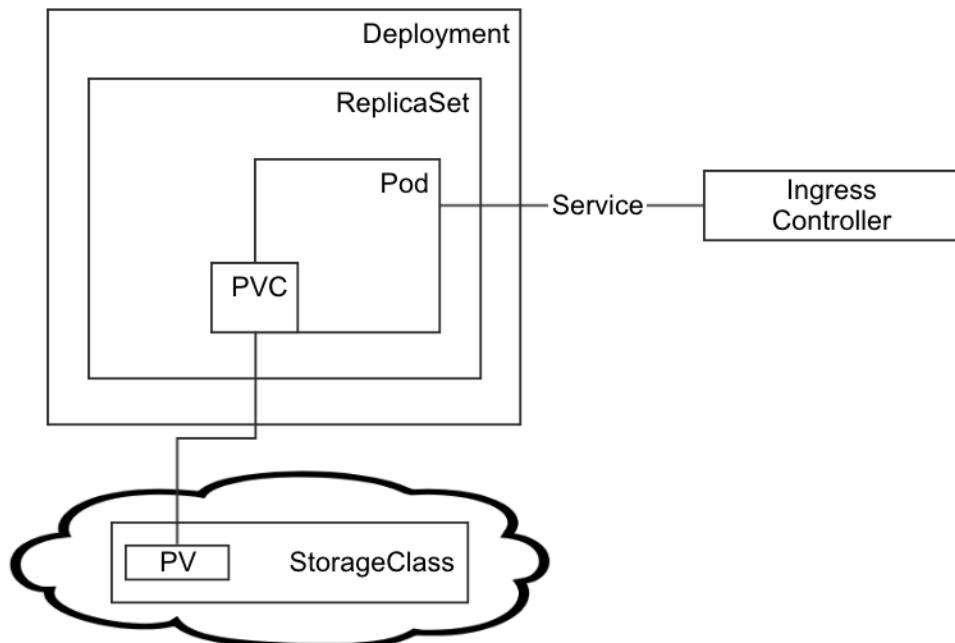
Outros componentes do Kubernetes que também merecem destaque são o *StorageClass* e o *Ingress*. De forma simples, uma *StorageClass* é uma classe que descreve como o armazenamento através de volumes persistentes (*Persistent Volume (PV)*) será provido a aplicações ([The Kubernetes Authors, 2018e](#)). Esta abordagem é vantajosa pois permite que diferentes *StorageClasses* com diferentes características possam ser criados no *cluster* Kubernetes, permitindo assim que a aplicação utilize a que melhor se adeque as suas necessidades. Para utilização de um volume é necessário que um *pod* solicite a criação deste volume através de uma requisição de volume persistente (*Persistent Volume Claim (PVC)*), a qual será processada pelo *Storage Class*.

Já o *Ingress* é um recurso do Kubernetes que atua na camada de aplicação da

arquitetura *Transmission Control Protocol (TCP)/IP* e permite a criação de regras de acesso a serviços do *cluster* através de domínios ou recursos personalizados, além de também possibilitar o balanceamento de carga e implementação de criptografia no acesso a estes serviços. Um *Ingress* consiste, portanto, de um conjunto de regras que permitem que o tráfego de entrada possa acessar os serviços do *cluster* (The Kubernetes Authors, 2018a). É importante ressaltar que aprimoramentos estão sendo realizados para possibilitar que a camada de transporte também seja controlada através do *Ingress*, mas esta funcionalidade ainda está em desenvolvimento (PLESHAKOV, 2018).

Outros objetos, tais como *ReplicaSets* e *ResourceQuotas* também são vastamente utilizados em *clusters* Kubernetes. Também é possível criar definições de recursos customizados (*Custom Resource Definition (CRD)*) através da *Application Program Interface (API)* do Kubernetes para que novos objetos possam ser utilizados. A Figura 5 exemplifica a organização dos objetos *Deployment*, *ReplicaSet*, *Pod*, *PVC*, *PV*, *StorageClass*, *Service* e *Ingress Controller* na implantação de uma aplicação em um *cluster* Kubernetes.

Figura 5 – Objetos do Kubernetes



Fonte: Elaborado pelo autor.

2.2.3.1.2 Helm

O Helm é um gerenciador de pacotes para Kubernetes (HELM, 2018). Estes pacotes são denominados *charts* e podem ser instalados em um *cluster* Kubernetes a partir de repositórios públicos ou privados, dispensando assim a necessidade de criar todos os recursos necessários para a implantação de uma aplicação manualmente.

Um *chart Helm* pode, por exemplo, implantar uma aplicação WordPress em um *cluster* de forma automatizada, criando de forma automatizada os *Deployments*, *Services* e *ReplicaSets* necessários para implantação da aplicação.

2.2.3.2 Rancher

Assim como o Kubernetes é uma ferramenta para orquestração de contêineres, o Rancher é uma plataforma para administração de *clusters* Kubernetes. Ele facilita, por exemplo, o monitoramento dos recursos de *clusters* e criação e manutenção de *clusters* híbridos ([RANCHER LABS, INC, 2018](#)).

A arquitetura em alto nível do Rancher é simples: é necessário apenas a criação de um servidor para gerenciamento dos *clusters* e configurar a conexão e permissões de acesso do Rancher a estes *clusters*. Também é possível utilizar a ferramenta de gerenciamento *Rancher Kubernetes Engine (RKE)* para rápida criação de *clusters* Kubernetes.

2.3 Sistemas de Arquivos

De acordo com [Tanenbaum \(2010\)](#), a parte do sistema operacional que trata do modo como os arquivos são estruturados, nomeados, acessados, usados, protegidos e implementados é o sistema de arquivos. Exemplos de sistemas de arquivos são o ISO 9660 (utilizado em CD-ROMs), FAT-32 (extensão do sistema de arquivos MS-DOS da Microsoft) e EXT4 (evolução do sistema de arquivos EXT3 do Linux).

2.3.1 Sistemas de Arquivos Distribuídos

Segundo [Neto et al. \(2017\)](#), um sistema de arquivos distribuído é um sistema no qual os arquivos estão armazenados e distribuídos em computadores diferentes interligados por meio de uma rede de comunicação. Um sistema de arquivos distribuído apresenta o mesmo conceito de um sistema de arquivos local, ou seja, ele também trata do gerenciamento de arquivos, porém no contexto distribuído. Ele permite, portanto, que usuários e programas remotos leiam e escrevam dados que estão aparentemente armazenados na máquina do usuário, mas que na verdade estão disponíveis remotamente. Isto significa, portanto, que ele cumpre a meta fundamental de ser um sistema transparente ao usuário.

Além das vantagens inerentes de rápida escalabilidade e maior redundância do sistema distribuído, sistemas de arquivos distribuídos também podem oferecer a vantagem de fornecer acesso a uma base unificada de dados aos usuários, evitando réplicas desnecessárias de arquivos em máquinas locais. Entretanto, uma desvantagem que merece destaque na utilização de sistemas de arquivos distribuídos é a necessidade de uma boa infraestrutura de rede para suportar o constante tráfego de arquivos, que pode ser ineficiente em uma rede precária e de baixas taxas de transmissão.

Exemplos de sistemas de arquivos distribuídos são o Ceph, GlusterFS, StorageOS, *Google File System (GFS)* e Lustre. Dos exemplos citados apenas o primeiro será objeto de estudo deste trabalho, pois a implementação deste sistema de arquivos em Kubernetes têm recebido grande apoio da [CNCF \(WATTS, 2018\)](#).

2.3.1.1 Ceph

Ceph é um sistema de armazenamento distribuído de código aberto. Este sistema de armazenamento permite que dados sejam armazenados não apenas no formato de arquivos, mas também como blocos e objetos ([Red Hat, Inc., 2016](#)).

O armazenamento através de arquivos ocorre quando os dados são salvos em uma única quantidade de armazenamento, geralmente organizada por diretórios. Já o armazenamento em blocos ocorre quando o armazenamento disponível é dividido em blocos e estes blocos podem ser utilizados por diferentes aplicações, seja de forma distribuída ou não. Finalmente, o armazenamento através de objetos é uma estrutura horizontal onde dados são armazenados como uma unidade denominada objeto e possuem um identificador único que localiza este objeto na estrutura de dados. Este identificador é salvo em um servidor de metadados, que também pode armazenar informações como data de criação do objeto e permissões de acesso, por exemplo ([Red Hat, Inc., 2018](#)).

O Ceph foi projetado sob a premissa de que grandes sistemas de armazenamento de dados possuem o potencial de armazenar ainda mais informações, e falhas de componentes não são exceções, e sim constantes ([MEYER; MORRISON, 2015](#)). Assumir que falhas de componentes são comuns no sistema e trabalhar para contornar estas falhas torna o Ceph um sistema poderoso.

Uma das características mais interessantes do Ceph é o algoritmo *Controlled Replication Under Scalable Hashing (CRUSH)*. O [CRUSH](#) é um algoritmo de locação pseudo-randômico que permite que clientes possam computar a localização de um dado no *cluster* sem precisar consultar uma tabela central que guarde a localização de todos os dados do *cluster*. Como consequência, o Ceph busca garantir a escalabilidade e confiabilidade do sistema através da computação inteligente dos dados nele armazenados ([GUDU; HARDT; STREIT, 2014](#)).

Cinco são os componentes principais de um *cluster* Ceph: *Object Storage Daemon (OSD)s*, *Monitors*, *Managers*, *Ceph Metadata Server (MDS)* e *RADOS*. Cada um destes componentes é explicado a seguir:

- [OSDs](#): um [OSD](#) armazena dados, processa replicação de dados e fornece informações de monitoramento para os *Monitors* e *Managers* ([Red Hat, Inc, 2016](#)). Estes são os componentes de “mais baixo nível” do *cluster* Ceph.

- *Monitors*: mantém mapas e informações do *cluster*, tais como mapa de outros *Monitors*, *OSDs* e *Managers* existentes, além do mapa CRUSH utilizado para computação da localização dos dados armazenados (Red Hat, Inc, 2016). *Monitors* também são responsáveis pela autenticação de clientes e *daemons*. Quando um cliente deseja armazenar ou acessar dados do *cluster* ele deve se conectar ao *Monitor* para se autenticar e obter o mapa do *cluster* e, após ser devidamente autenticado, ele poderá conectar-se diretamente o *OSD* correspondente para realizar a operação desejada (Inktank Storage, Inc, 2014).
- *Managers*: os *Managers* são responsáveis por registrar métricas de execução e estado do *cluster* Ceph, além da carga de trabalho do sistema. Os *Managers* permitem a consulta de métricas do sistema através de *plugins* como uma *dashboard* (painele de controle) e uma *Representational State Transfer* (REST) API (Red Hat, Inc, 2016).
- *MDSs*: armazenam metadados do sistema de arquivos Ceph, como permissões e propriedades de arquivos, ou seja, se o Ceph for utilizado como armazenamento em blocos ou objetos *MDSs* não serão necessários (Red Hat, Inc, 2016).
- *RADOS*: consiste na junção de *OSDs* e *Monitors* para distribuição de dados como objetos pelo *cluster* e replicação de dados (Openstack, 2014). O *RADOS Gateway* (RGW) é uma interface que fornece uma REST API para acesso aos objetos armazenados pelo *RADOS*. Esta API é compatível com as interfaces S3 da Amazon e OpenStack Swift (Inktank Storage, Inc, 2012).

2.3.1.2 Rook

O Rook é um projeto de código aberto incubado pela CNCF (WATTS, 2018). De forma simples, o Rook é um orquestrador de armazenamento para *clusters* Kubernetes que automatiza a implantação do Ceph nestes sistemas. No presente momento (segundo semestre de 2018) a implementação do Ceph através do Rook está em estado Beta (ROOK, 2018a).

O Rook permite, por exemplo, que sejam automaticamente implantados todos os *OSDs* necessários em nodos específicos, dispensando assim a necessidade de manualmente instalar e configurar os *daemons* necessários nestes nodos. Assim como o Ceph, é possível utilizar o Rook para orquestrar o armazenamento de dados através de arquivos, blocos e objetos.

O armazenamento através de blocos pode ser empregado no Rook através da implantação de uma *Storage Class* do Rook via CRD. Já o armazenamento de objetos pode ser realizado através de uma *Object Store*, a qual - assim como o *RADOS* do Ceph - oferece uma REST API para acesso aos objetos armazenados. Já um sistema de arquivos

distribuído pode ser implantado no Rook através de um *Shared File System*, o qual também pode ser associado a serviços como um volume.

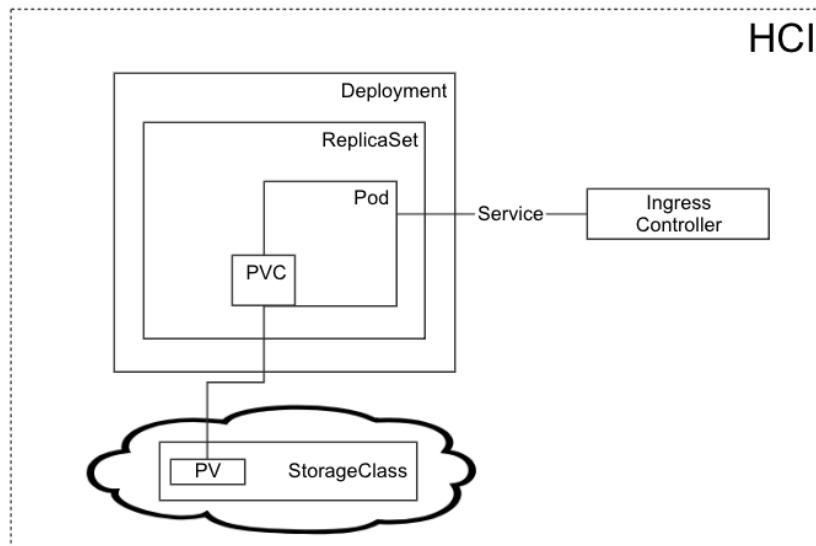
A maior parte dos componentes criados pelo Rook no *cluster* são componentes equivalentes aos do Ceph. Um *pod* [OSD](#) do Rook, por exemplo, também possui como função armazenar os dados em disco e prover acesso a estes dados para outros programas.

2.3.2 Convergência de infraestrutura

Uma Infraestrutura Convergente ([CI](#)) é a convergência das infraestruturas de computação, armazenamento e rede em um *data center* ([NETAPP, 2018](#)). A [CI](#) almeja alcançar o melhor desempenho do sistema com padronização dos protocolos e *hardware* utilizado no *data center*, muitas vezes através da utilização de *hardware* proprietário que realize funções específicas ([NETAPP, 2018](#)).

Uma Infraestrutura Hiperconvergente ([HCI](#)) é uma infraestrutura definida por *software* que pode ser dimensionada horizontalmente ([VMWARE, 2018](#)) e que combina computação, virtualização, armazenamento e rede em um único *cluster* ([CISCO, 2018](#)). A hiperconvergência traz simplicidade a *clusters* locais com uma única e simples plataforma de computação e gerenciamento. Com a [HCI](#), todas as principais funções do *data center* são executadas em uma camada de *software* altamente integrada, simplificando o fornecimento de serviços que antes exigiam *hardware* específico para uma finalidade. Dentre as vantagens oferecidas pela hiperconvergência encontram-se a simplicidade de administração do sistema e o baixo custo total de propriedade ([VMWARE, 2018](#)). Como a proposta de uma infraestrutura hiperconvergente consiste na combinação das infraestruturas de computação, virtualização, armazenamento e rede em um único *cluster*, é possível alcançar este modelo através de uma infraestrutura de contêineres, que pode combinar todas as infraestruturas através de uma única plataforma de gerenciamento. A [Figura 6](#) mostra que armazenamento, rede e computação podem ser combinados através da utilização de contêineres, sendo o armazenamento implementado através do *StorageClass*, a rede através de *Services* e recursos *Ingress* (gerenciados pelo *Ingress Controller*) e a computação através dos *Pods*, *Deployments* e *ReplicaSets*.

Figura 6 – Infraestrutura hiperconvergente de contêineres



Fonte: Elaborado pelo autor.

3 IMPLEMENTAÇÃO DO SISTEMA DE ARMAZENAMENTO DISTRIBUÍDO

A proposta de implementação deste trabalho consiste na criação de um *cluster* Kubernetes e implantação de armazenamento distribuído utilizando a plataforma de armazenamento Rook.

O *cluster* Kubernetes foi criado através da utilização de três máquinas virtuais e uma sub-rede configuradas na [GCP](#). A infraestrutura foi criada remotamente na [GCP](#) utilizando o Terraform, que é uma aplicação para manutenção e gerenciamento de infraestruturas através de código. Escolheu-se a [GCP](#) para implementação do trabalho devido ao número de créditos de teste disponíveis para utilização na plataforma, mas outros provedores de infraestrutura tais como [AWS](#) ou *Microsoft Azure* poderiam ter sido utilizados já que o Terraform também suporta a criação de infraestruturas nestes provedores. É importante ressaltar que foram utilizadas máquinas virtuais para que o cenário implementado pudesse ser replicado tanto em outros provedores quanto localmente, não tornando assim a implementação dependente de [APIs](#) ou serviços específicos do provedor, tais como [AKS](#) ou [Amazon EKS](#).

Já o *cluster* Kubernetes foi criado através do Rancher, que é uma ferramenta para gerenciamento de *clusters* Kubernetes. Esta ferramenta permite, por exemplo, a criação automatizada de *clusters* e adição e remoção de nós de processamento ao *cluster*, além de fornecer ferramentas para monitoramento dos serviços implantados. O Rancher já é utilizado no ambiente de produção da infraestrutura de serviços mantida no câmpus do [IFSC](#) de São José.

Após a criação da infraestrutura necessária e implantação do *cluster* Kubernetes, o Rook foi implantado para gerir o armazenamento do *cluster*, e uma base de dados MySQL e um *blog* WordPress foram instalados para utilização do armazenamento fornecido pelo Rook. Instruções detalhadas sobre a criação da infraestrutura remota e do *cluster* Kubernetes estão disponíveis na seção de Anexos.

3.1 Criação da infraestrutura remota

As especificações das máquinas virtuais utilizadas, tais como tipo de máquina utilizada e distribuição Linux instalada estão disponíveis na tabela [Tabela 1](#).

A sub-rede criada possui regras de *firewall* específicas para permitir que apenas

Tabela 1 – Especificação das máquinas virtuais

Especificação	Valor
Tipo de máquina virtual	<i>n1-standard-2</i>
Número de máquinas virtuais criadas	3
Número de <i>cores</i>	2 <i>Virtual Central Processing Units (vCPUs)</i>
Quantidade de memória RAM	7.5 GB
Tipo de disco rígido	<i>Standard persistent disk</i>
Quantidade de armazenamento	15 GB
Zona e Região	<i>southamerica-east1-a</i>
Distribuição Linux	Debian 9 (<i>Stretch</i>)

Fonte – Elaborado pelo autor.

as portas necessárias para funcionamento dos serviços implantados e gerenciamento do *cluster* Kubernetes estejam abertas. A tabela Tabela 2 mostra quais portas foram abertas para os protocolos **TCP** e *User Datagram Protocol (UDP)*.

Tabela 2 – Especificação da sub-rede

Especificação	Valor
Faixa de IPs	10.0.0.0/24
Tipo de endereço	Interno
Portas de <i>firewall</i> abertas para protocolo TCP	22, 53, 80, 443, 2380, 2379, 3306, 6443, 6970, 6800-7300, 8124, 9283, 10250, 10254, 30000-32767, 44134
Portas de <i>firewall</i> abertas para protocolo UDP	53, 8472

Fonte – Elaborado pelo autor.

A criação da infraestrutura remota foi realizada utilizando o Terraform, que permite que componentes da infraestrutura sejam descritos como código e gerenciados através do *software*. O Código 3.1 mostra um trecho do código utilizado para criação de uma das máquinas virtuais utilizadas no projeto através do Terraform.

Código 3.1 – Configuração de máquina virtual descrita no arquivo de configuração do Terraform

```
resource "google_compute_instance" "vm-0" {
  name        = "vm-0"
  machine_type = "n1-standard-2"
  zone        = "southamerica-east1-a"
```

```

boot_disk {
  initialize_params {
    image = "debian-cloud/debian-9"
    size = "15"
  }
}

network_interface {
  subnetwork      = "${google_compute_subnetwork.subnet-0.name}"
  address        = "${google_compute_address.address-0.address}"
  access_config  = {}
}

metadata {
  sshKeys = "${var.gce_ssh_user}:${file(var.gce_ssh_pub_key_file)}"
}

metadata_startup_script = "${data.template_file.startup_script.rendered}"

tags = ["k8s"]
}

```

A sintaxe da configuração é: o recurso criado é do tipo `google_compute_instance` (máquina virtual) e nomeado como `vm-0`. A máquina virtual é do tipo predefinido `n1-standard-2` e localizada na zona e região `southamerica-east1-a` (São Paulo). O disco de inicialização da máquina terá armazenamento de 15 GB e será instalada a imagem da distribuição linux `debian-cloud/debian-9`. O nome da rede ao qual a máquina será associada está definido no parâmetro `google_compute_subnetwork.subnet-0.name` e endereço de rede interno será o definido no parâmetro `google_compute_address.address-0.address`. O nome do usuário criado na máquina virtual será o armazenado na variável `var.gce_ssh_user` e suas chaves para acesso através *Secure Shell (SSH)* estão disponíveis no arquivo referenciado pela variável `var.gce_ssh_pub_key_file`. Também será adicionado um `script` de inicialização a ser executado toda vez que a máquina for iniciada, o qual está definido em `data.template_file.startup_script.rendered`. Este `script` está disponível no [Apêndice A](#). Também será adicionada à máquina virtual uma `tag` de valor `k8s`. O arquivo completo utilizado para criação de toda a infraestrutura está disponível no [Apêndice B](#).

Após a descrição dos componentes da infraestrutura necessários no formato aceito pelo Terraform a infraestrutura remota foi criada na [GCP](#). A [Figura 7](#) mostra as máquinas virtuais criadas através do Terraform na [GCP](#).

Figura 7 – Máquinas virtuais criadas na GCP

The screenshot shows the Google Cloud Platform interface for managing VM instances. At the top, there are buttons for 'CREATE INSTANCE', 'IMPORT VM', 'REFRESH', 'START', and 'STOP'. Below this is a search bar labeled 'Filter VM instances'. A table lists three VM instances:

	Name	Zone	Recommendation	Internal IP	External IP	Connect
<input type="checkbox"/>	vm-0	southamerica-east1-a		address-0 (10.0.0.100) (nic0)	35.198.47.124	SSH ▼ ⋮
<input type="checkbox"/>	vm-1	southamerica-east1-a		address-1 (10.0.0.101) (nic0)	35.198.29.170	SSH ▼ ⋮
<input type="checkbox"/>	vm-2	southamerica-east1-a		address-2 (10.0.0.102) (nic0)	35.199.81.13	SSH ▼ ⋮

Fonte: Elaborado pelo autor.

3.2 Criação do *cluster* Kubernetes

Para criação do *cluster* Kubernetes remoto foi utilizado o Rancher, que é uma ferramenta para gerenciamento de *clusters* Kubernetes. Os únicos requisitos para criação do *cluster* remoto utilizando o Rancher são de que os dispositivos que farão parte do *cluster* possuam versões do Docker instalado compatíveis com a versão do Rancher utilizada e que sejam fornecidas ao Rancher chaves SSH de um usuário que possua privilégios de administrador. Assim, os componentes necessários, como contêineres, serão automaticamente instalados e o *cluster* Kubernetes criado. Escolheu-se utilizar o Rancher por dois motivos principais, sendo o primeiro a vantajosa automatização da criação do *cluster* (que pode ser complexa devido as configurações de rede) e o segundo a independência de plataforma que o Rancher oferece, podendo ser utilizado para gerenciar tanto um *cluster* remoto hospedado em diferentes provedores de IaaS/PaaS ou local.

Após a instalação do docker nas máquinas virtuais (procedimento automaticamente realizado na criação das máquinas virtuais através de um *script* de inicialização) é necessário dar permissão de execução do comando docker ao usuário na máquina virtual. Assim, as credenciais de acesso deste usuário poderão ser utilizadas pelo Rancher para criar e configurar o *cluster* Kubernetes remoto através do docker instalado nas máquinas. Parte do arquivo de configuração do Rancher utilizado nesta etapa é mostrado no [Código 3.2](#), enquanto o arquivo completo de configuração pode ser visto no [Apêndice C](#).

Código 3.2 – Configuração de nó mestre do *cluster* criado com Rancher

```
nodes:
- address: 35.198.47.124
  internal_address: 10.0.0.100
  user: matuzalemmuller
  ssh_key_path: keys/id_rsa
```

```

role: [controlplane,worker,etcd]
hostname_override: master
labels:
  node: master
}

```

Uma breve explicação sobre cada parâmetro utilizado na configuração está disponível abaixo:

- **address**: endereço IP público da máquina virtual.
- **internal_address**: endereço IP da máquina virtual na sub-rede na qual ela se encontra.
- **user**: usuário que será utilizado para criação do *cluster*. É necessário que este usuário possua permissões de execução do docker e outras permissões de administrador para que o *cluster* seja criado.
- **ssh_key_path**: localização das chaves SSH do usuário utilizado no parâmetro **user** (neste caso, matuzalem@muller).
- **role**: funções do nó no *cluster*. Como este será o nó mestre ele assumirá as funções de **controlplane** (*master*), **worker** (executa *pods*) e **etcd** (armazena as chaves-valor (*key-value*) e estado de aplicações de forma distribuída).
- **hostname_override**: ignora o nome da máquina virtual e utiliza um nome específico (**master**) para se referenciar a este nó.
- **labels**: é adicionado um **label** de valor **master** para que *pods* possam ser criados especificamente neste nó ao referenciar este **label**, por exemplo.

A [Figura 8](#) mostra o *cluster* disponível após sua criação com os três nós do *cluster* Kubernetes em execução. A ferramenta de linha de comando **kubectl** é utilizada para gerenciamento do *cluster* Kubernetes.

Figura 8 – Saída do terminal: Nós em execução após criação de *cluster* com Rancher

```

matuzalem-macbook:remote-setup matuzalem$ kubectl get node
NAME      STATUS    ROLES                  AGE      VERSION
master    Ready     controlplane,etcd,worker  1m       v1.11.1
worker1   Ready     etcd,worker            1m       v1.11.1
worker2   Ready     etcd,worker            1m       v1.11.1

```

Fonte: Elaborado pelo autor.

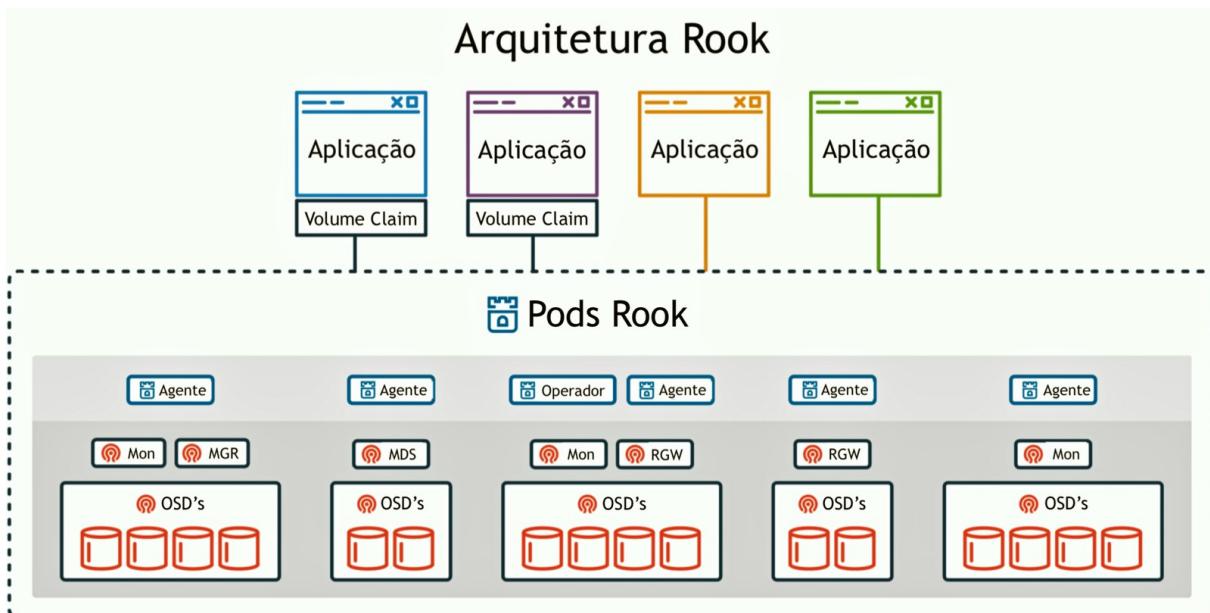
Após a criação do *cluster* um arquivo de nome `kube_config_cluster.yml` será criado. Este arquivo de configuração permite que o *cluster* seja acessado e gerenciado externamente.

3.3 Implantação do Rook para utilização de armazenamento

Após a criação da infraestrutura e do *cluster* Kubernetes é possível instalar o Rook para que seja realizada a gestão de armazenamento no *cluster*.

A instalação do Rook consiste em três procedimentos principais, sendo o primeiro a instalação do Rook no *cluster* para criação de operadores e agentes nos nós. Isto permitirá que o Ceph seja iniciado nos nós, mas nenhum disco será mapeado ainda. Para que os discos sejam mapeados pelo Rook (e consequentemente pelo Ceph) é necessário criar o *cluster* Rook. Este *cluster* irá criar *pods* adicionais em cada nó, que consistem em *monitors* e *OSDs*, além do *operator* criado no nó mestre. Um *operator* é, de forma simples, um recurso utilizado para permitir que sejam implantados CRDs e ciclos de automatização de tarefas ao implantar aplicações, não sendo necessário criar estas funcionalidades ao implantar as aplicações (COREOS, 2018). Após a instalação do Rook e criação do *cluster* para mapeamento dos discos é possível utilizar o Rook para prover armazenamento. A Figura 9 demonstra uma arquitetura do Rook que contempla todos os tipos de armazenamento disponíveis e os componentes utilizados para prover o armazenamento.

Figura 9 – Representação da arquitetura do Rook



Fonte: Traduzido de (ROOK, 2018b).

Os componentes criados pelo Rook são diretamente equivalentes aos componentes do Ceph. Isto significa, por exemplo, que um *monitor* do Rook é a implementação do

monitor do Ceph no *cluster* Kubernetes. O mesmo é valido para Agentes, Operadores, **OSDs**, **MDSs** e **RGWs**. Já as *Volume Claims* representam requisições das aplicações para o *cluster*, requisitando um volume para armazenamento de dados.

Em termos de implementação, o Rook *operator*, que consiste nos operadores e agentes do Rook, pode ser instalado no *cluster* através do Helm. Neste trabalho a versão utilizada do Rook foi a *release* versão 0.8.0, a qual possui a implementação do Ceph em estado beta. Como há um mestre e dois nós no *cluster* Kubernetes, serão criados no *namespace rook-ceph-system* três *pods* de *agents*, um em cada nó. Além destes *pods* também serão criados *discoverers* (responsáveis por identificar os dispositivos disponíveis para armazenamento nas máquinas virtuais) e um *operator*, sendo o último disponível apenas no nó mestre. A [Figura 10](#) mostra os componentes criados no *namespace rook-ceph-system*.

Figura 10 – Saída do terminal: *Pods* criados pelo Rook *Operator*

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-agent-j7vkn	1/1	Running	0	1m
rook-ceph-agent-lgzqm	1/1	Running	0	1m
rook-ceph-agent-sngjz	1/1	Running	0	1m
rook-ceph-operator-59c6b78b7c-1sz2l	1/1	Running	0	1m
rook-discover-67t94	1/1	Running	0	1m
rook-discover-b69j4	1/1	Running	0	1m
rook-discover-ljzqv	1/1	Running	0	1m

Fonte: Elaborado pelo autor.

Após realizar a inicialização do Rook *operator* é possível criar o *cluster* Rook. A criação do *cluster* foi manualmente realizada utilizando o código disponível no [Apêndice D](#). Este código cria o *cluster*, *namespace* e outros componentes necessários para a utilização do Rook. Neste caso os principais componentes criados serão um *pod manager*, três *pods* de *monitors* e três *pods* de **OSDs**, os quais são criados no *namespace rook-ceph*. A [Figura 11](#) mostra os pods criados em execução no *cluster* Rook.

Agora que o *cluster* Rook foi criado é possível configurá-lo para utilizá-lo para prover o armazenamento de objetos, blocos ou arquivos. Neste trabalho os tipos de armazenamento utilizados serão blocos e objetos.

O armazenamento em blocos será utilizado para gestão de volumes persistentes a serem utilizados por contêineres, os quais persistem dados após seu reinício. Isto permite, por exemplo, que os contêineres não percam suas configurações ou estados. Esta é uma boa forma de armazenar configurações e conteúdos que são frequentemente modificados por contêineres. Já o armazenamento através de objetos será utilizado para armazenar e distribuir conteúdos estáticos, tais como imagens e arquivos que são pouco modificados.

Figura 11 – Saída do terminal: *Pods* criados pelo *Cluster Rook*

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-mgr-a-7944d8d79b-sjfxw	1/1	Running	0	3m
rook-ceph-mon0-5qj71	1/1	Running	0	4m
rook-ceph-mon1-rfvpj	1/1	Running	0	4m
rook-ceph-mon2-xc966	1/1	Running	0	4m
rook-ceph-osd-id-0-6cdfc557f-n6hdk	1/1	Running	0	3m
rook-ceph-osd-id-1-5475fbb74-wbh74	1/1	Running	0	3m
rook-ceph-osd-id-2-5f68f97d4d-87hpx	1/1	Running	0	3m
rook-ceph-osd-prepare-master-7dbhx	0/1	Completed	0	3m
rook-ceph-osd-prepare-worker1-cwzf7	0/1	Completed	0	3m
rook-ceph-osd-prepare-worker2-cqbc2	0/1	Completed	0	3m

Fonte: Elaborado pelo autor.

O armazenamento em blocos pode ser implementado no Rook através de um *StorageClass*, enquanto o armazenamento de objetos é disponibilizado através de uma *Object Store*. Ambos serão explicados nos parágrafos seguintes.

Para a configuração do armazenamento de dados através de objetos é necessário criar um *StorageClass* específico do Rook. A criação deste *StorageClass* é realizado utilizando o código disponível no [Apêndice E](#). Após a criação deste *StorageClass* volumes persistentes (**PV**) poderão ser criados através de requisições de volumes (**PVC**) ao Rook.

Já para o armazenamento de objetos no *cluster* Rook é necessário que uma *Object Store* seja criada. Uma *Object Store* é um [CRD](#) criado pelo Rook e possui como função criar os componentes necessários para gerenciamento dos objetos no RADOS do Ceph, fornecendo assim acesso a estes objetos através de um *pod* equivalente ao [RADOS Gateway](#). Assim como o [RADOS Gateway](#) oferece uma [REST API](#) de acesso compatível com a interface S3 da Amazon aos objetos armazenados, o *cluster* Rook também permite que arquivos sejam gerenciados através desta [API](#). Logo, é possível criar *buckets* no *cluster* Rook, os quais serão utilizados para armazenar dados estáticos e publicamente acessíveis. Neste trabalho um domínio .cc foi utilizado para validação do modelo, utilizando assim o balanceamento de carga de requisições *Hypertext Transfer Protocol* ([HTTP](#)) também através de *Domain Name System* ([DNS](#)).

A [Figura 12](#) mostra o *pod* criado para execução do [RADOS gateway](#) no *cluster* Rook, juntamente com os outros *Pods* disponíveis no namespace `rook-ceph`.

Figura 12 – Saída do terminal: *Pod RADOS Gateway* em execução

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-mgr-a-7944d8d79b-sjfxw	1/1	Running	0	10m
rook-ceph-mon0-5qj71	1/1	Running	0	11m
rook-ceph-mon1-rfvpj	1/1	Running	0	11m
rook-ceph-mon2-xc966	1/1	Running	0	10m
rook-ceph-osd-id-0-6cdfc557f-n6hdk	1/1	Running	0	10m
rook-ceph-osd-id-1-5475fbb74-wbh74	1/1	Running	0	10m
rook-ceph-osd-id-2-5f68f97d4d-87hpx	1/1	Running	0	10m
rook-ceph-osd-prepare-master-7dbhx	0/1	Completed	0	10m
rook-ceph-osd-prepare-worker1-cwzf7	0/1	Completed	0	10m
rook-ceph-osd-prepare-worker2-cqbc2	0/1	Completed	0	10m
rook-ceph-rgw-my-store-fdd764948-vb7fx	1/1	Running	0	13s

Fonte: Elaborado pelo autor.

3.4 Implantação de serviços para utilizar armazenamento

Para utilização do armazenamento foram implantados uma base dados MySQL e um *blog* WordPress. A base de dados é necessária para o funcionamento do *blog* WordPress. A base de dados MySQL utiliza armazenamento em blocos na forma de volumes, enquanto o *blog* WordPress utiliza armazenamento tanto na forma de objetos quanto em blocos. A seguir serão descritas as implementações destes serviços na nuvem remota.

3.4.1 MySQL

A implementação da base de dados MySQL foi realizada através da instalação do serviço pelo Helm. Como é utilizado um gerenciador de pacotes para realizar a instalação da base de dados a sua implantação é rápida e automatizada.

Para utilização do armazenamento em blocos o *pod* MySQL realiza uma **PVC** ao Rook, ou seja, o *pod* solicita a criação de um volume de dados persistentes (**PV**) ao Rook para que este volume seja utilizado como um bloco de armazenamento. O Rook, por sua vez, fornece ao *pod* MySQL o volume requisitado. A [Figura 13](#) mostra a requisição do volume (**PVC**) realizada pelo *pod* MySQL e o volume (**PV**) entrege ao *pod* pelo Rook.

A [Figura 14](#) mostra o *pod* MySQL em execução, poucos momentos após sua criação.

3.4.2 WordPress

A implementação do WordPress também foi realizada através do Helm, ou seja, o serviço foi rapidamente criado no *cluster* Kubernetes e o Rook foi utilizado para gerir

Figura 13 – Saída parcial do terminal: Requisição de volume e volume utilizado pelo *pod* MySQL

```
matuzalem-macbook:tcc-engtelecom matuzalem$ kubectl get pvc
NAME      STATUS      VOLUME                                     CAPACITY
mysql     Bound       pvc-c3428e0b-f66b-11e8-939f-42010a000064   5Gi
matuzalem-macbook:tcc-engtelecom matuzalem$ kubectl get pv
NAME                                     CAPACITY  STORAGECLASS
pvc-c3428e0b-f66b-11e8-939f-42010a000064  5Gi        rook-ceph-block
```

Fonte: Elaborado pelo autor.

Figura 14 – Saída do terminal: *Pod* MySQL em execução

```
matuzalem-macbook:tcc-engtelecom matuzalem$ kubectl get pod
NAME                  READY   STATUS    RESTARTS   AGE
mysql-56655cb669-2zgvc   1/1     Running   0          3m
```

Fonte: Elaborado pelo autor.

o armazenamento utilizado pelo WordPress. Assim como no *pod* MySQL, um volume também foi criado e associado ao *pod* WordPress através de uma **PVC**. Ambos volume e requisição são mostrados na [Figura 15](#), juntamente com o volume e requisição do serviço MySQL.

Figura 15 – Saída parcial do terminal: Requisição de volume e volume utilizado pelo *pod* WordPress

```
matuzalem-macbook:tcc-engtelecom matuzalem$ kubectl get pvc
NAME      STATUS      VOLUME                                     CAPACITY
mysql     Bound       pvc-c3428e0b-f66b-11e8-939f-42010a000064   5Gi
wordpress  Bound       pvc-f035bafe-f66c-11e8-939f-42010a000064   5Gi
matuzalem-macbook:tcc-engtelecom matuzalem$ kubectl get pv
NAME                                     CAPACITY  STORAGECLASS
pvc-c3428e0b-f66b-11e8-939f-42010a000064  5Gi        rook-ceph-block
pvc-f035bafe-f66c-11e8-939f-42010a000064  5Gi        rook-ceph-block
```

Fonte: Elaborado pelo autor.

A [Figura 16](#) mostra que o *pod* WordPress também está em execução. É possível perceber que há quatro *pods* em execução no *namespace default*: um do MySQL, outro do WordPress e dois referentes ao NGINX *controller*. Os *pods* do NGINX funcionam como controladores *Ingress* para que o serviço WordPress possa ser acessado externamente. O NGINX pode ser instalado através do Helm e a interação entre os serviços NGINX

e WordPress ocorre automaticamente, ou seja, ao instalar o WordPress ele pode ser automaticamente configurado para ser acessado através do NGINX.

Figura 16 – Saída do terminal: *Pod* WordPress em execução

NAME	READY	STATUS	AGE
mysql-56655cb669-2zgvc	1/1	Running	13m
nginx-ingress-controller-7dd6474678-5jkqs	1/1	Running	24m
nginx-ingress-default-backend-5b88698ddc-zc242	1/1	Running	24m
wordpress-wordpress-5fb9cd9cf9-nr7rt	1/1	Running	5m

Fonte: Elaborado pelo autor.

A Figura 17 mostra a página web do *blog* WordPress em execução.

Figura 17 – *Blog* WordPress



Fonte: Elaborado pelo autor.

Até esta etapa o Rook está sendo utilizado para prover armazenamento através de volumes requisitados pelos serviços implantados. Isto significa que o armazenamento utilizado é em blocos, e o armazenamento de objetos ainda não foi utilizado.

Para utilizar o armazenamento de objetos da forma proposta é necessário armazenar objetos na *Object Store* do Rook e tornar estes objetos publicamente acessíveis. Neste trabalho uma imagem será salva como um objeto na *Object Store* do Rook e ela será utilizada pelo WordPress no *blog*. Para isto, é necessário criar um usuário no **RADOS gateway** e utilizar este usuário para criar um *bucket* e administrar o armazenamento dos objetos. Após a criação do usuário e *bucket* é possível armazenar uma imagem como objeto

na *Object Store*. A imagem utilizada neste projeto é a da [Figura 18](#). Esta imagem não possui direitos autorais, é do tipo jpg e possui 179 KB.

Figura 18 – Imagem armazenada como objeto



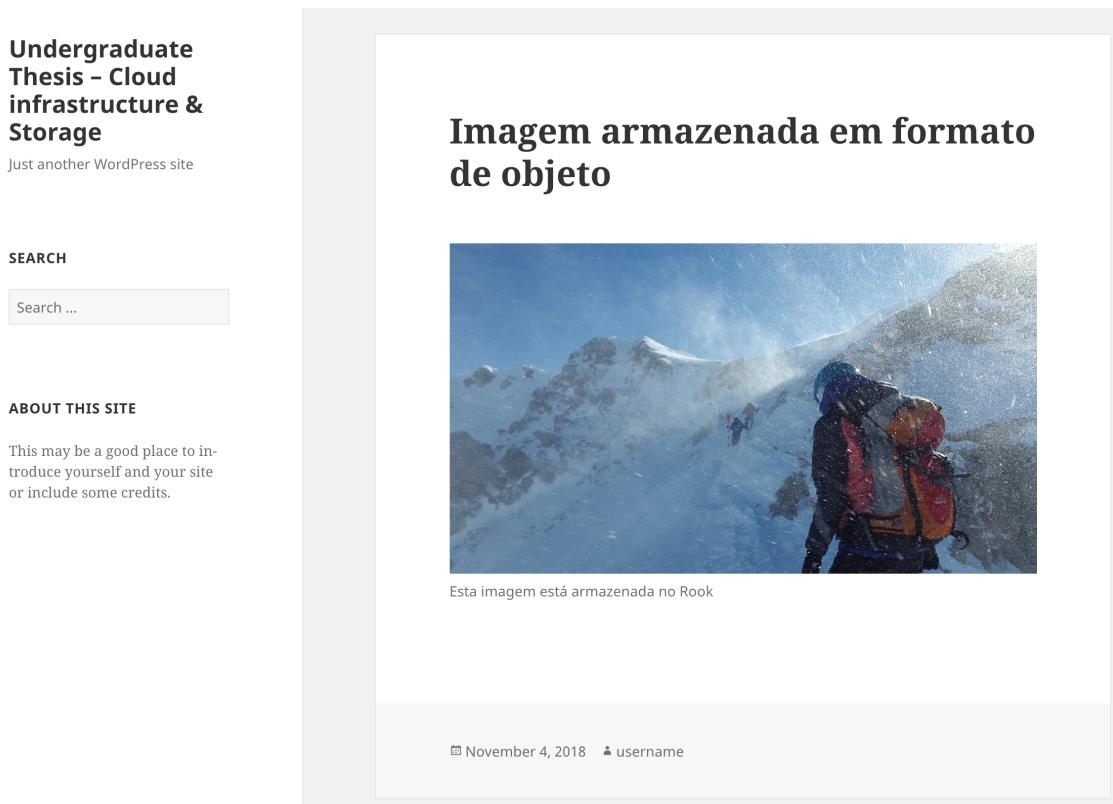
Fonte: ([STEINBERGER, 2017](#))

Agora que a imagem está armazenada como um objeto é necessário torna-lá pública, o que pode ser feito através da utilização do *Ingress NGINX*. O código utilizado para implantar o recurso *ingress* está disponível no [Apêndice F](#).

Com a imagem publicada é possível utilizá-la no WordPress. A [Figura 19](#) mostra a utilização da imagem em uma publicação no WordPress.

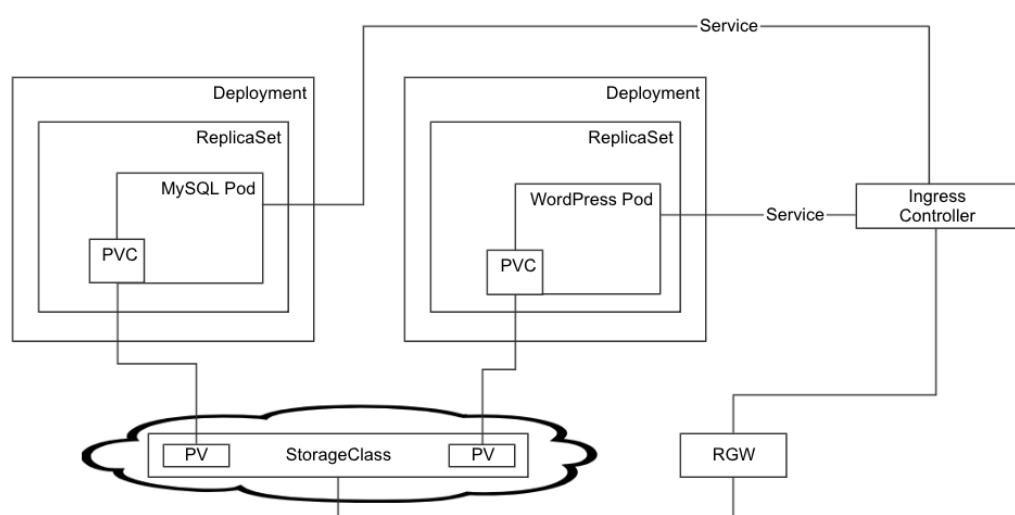
A [Figura 20](#) é um diagrama geral da implementação realizada neste trabalho. Os objetos `Deployment`, `ReplicaSet`, `MySQL Pod` e `WordPress Pod` são automaticamente criados através da instalação de *charts* disponíveis no Helm. As `PVC` são realizadas ao `StorageClass`, que cria volumes (`PV`) a serem utilizados pelos `Pods`. Já o objeto `RGW` é criado através da *Object Store* do Rook. É importante ressaltar que ambos o `StorageClass` e `RGW` são `CRDs` do Rook e utilizam o Ceph para armazenamento dos dados. O `Ingress Controller`, por sua vez, é configurado por objetos *Ingress* automaticamente criados pelas aplicações. A automatização na implantação de serviços e gerenciamento dos recursos comprova a facilidade de utilização do Kubernetes, além de sua eficácia em fornecer uma única plataforma de gerenciamento que combina rede, armazenamento e computação, caracterizando assim este modelo como uma infraestrutura hiperconvergente.

Figura 19 – Publicação WordPress utilizando imagem armazenada como objeto



Fonte: Elaborado pelo autor.

Figura 20 – Esquemático da implementação



Fonte: Elaborado pelo autor.

4 CONSIDERAÇÕES FINAIS

A automatização da criação da infraestrutura remota e seu gerenciamento através de código utilizando o Terraform facilitou o desenvolvimento do trabalho. Com o Terraform é possível não apenas adicionar componentes da nuvem pública utilizada, mas também aprimorar sua configuração ou removê-los da infraestrutura. A abstração fornecida pelo gerenciamento dos recursos através de código é interessante não apenas pela automatização, mas também por possibilitar a adoção de um padrão de versionamento de infraestrutura.

A utilização do Kubernetes como orquestrador de contêineres do *cluster* também foi interessante devido a automatização de tarefas oferecida. É possível, por exemplo, configurar a escalabilidade do serviço implantado através de replicações automáticas de *pods* caso a carga de trabalho aumente. Assim, se a carga de trabalho do *pod* WordPress aumentar, é possível que o orquestrador automaticamente crie novos *pods* para distribuir a carga de trabalho. Além disso, caso ocorra a falha de um *pod*, o Kubernetes irá automaticamente reiniciar este *pod*, montando novamente os volumes anteriormente utilizados e gerenciados pelo Rook. O orquestrador possibilita, portanto, que serviços sejam automaticamente reinicializados caso necessário, dispensando assim a intervenção direta do administrador para tratar falhas mais simples de serviços.

O Rook, por sua vez, mostrou ser uma alternativa interessante para o fornecimento de armazenamento distribuído em uma infraestrutura hiperconvergente de contêineres. A possibilidade de oferecer armazenamento através de blocos, arquivos e objetos é um ponto positivo, possibilitando que o administrador do serviço possa escolher o tipo de armazenamento que melhor se adeque às suas necessidades. Ao utilizar o cliente da *Amazon S3* para acessar e gerenciar o *bucket* criado no Rook foi possível perceber a similaridade entre os serviços, o que também é um ponto positivo já que a *Amazon S3* é um serviço de armazenamento consolidado no mercado. Esta similaridade permite, por exemplo, que dados que são armazenados em um *cluster* Rook sejam posteriormente transferidos para a *Amazon S3*, ou vice-versa.

4.1 Trabalhos futuros

Sugerem-se os seguintes temas para trabalhos futuros:

- Estudo de necessidades de armazenamento da rede do IFSC
- Proposta de implantação de sistema de armazenamento distribuído baseado em contêineres no IFSC

- Proposta de implantação de infraestrutura hiperconvergente no IFSC

REFERÊNCIAS

- AHN, J. et al. *Leveraging Containers and OpenStack*: A comprehensive review. 2018. Disponível em: <<https://www.openstack.org/containers/leveraging-containers-and-openstack/>>. Acesso em: 03 de dezembro de 2018. Citado na página 37.
- ARINZE, B.; ANANDARAJAN, M. Adapting cloud computing service models to subscriber requirements. In: *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. [S.l.: s.n.], 2013. p. 1–5. ISSN 1347-6890. Citado 2 vezes nas páginas 32 e 33.
- AWS. *What is Cloud Object Storage?* 2018. Disponível em: <<https://aws.amazon.com/what-is-cloud-object-storage/>>. Acesso em: 24 de novembro de 2018. Citado na página 29.
- BARI, M. F. et al. Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, v. 15, n. 2, p. 909–928, Second 2013. ISSN 1553-877X. Citado 3 vezes nas páginas 35, 36 e 37.
- BARZU, A. P.; CARABAS, M.; TAPUS, N. Scalability of a web server: How does vertical scalability improve the performance of a server? In: *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. [S.l.: s.n.], 2017. p. 115–122. Citado na página 36.
- BELBERGUI, C.; ELKAMOUN, N.; HILAL, R. Cloud computing: Overview and risk identification based on classification by type. In: *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*. [S.l.: s.n.], 2017. p. 1–8. Citado 2 vezes nas páginas 31 e 32.
- CISCO. *What Is Hyperconverged Infrastructure?* 2018. Disponível em: <<https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-hyperconverged-infrastructure.html>>. Acesso em: 24 de novembro de 2018. Citado na página 45.
- COREOS. *Operators*. 2018. Disponível em: <<https://coreos.com/operators/>>. Acesso em: 04 de dezembro de 2018. Citado na página 52.
- CTIC - IFSC câmpus São José. *Repositórios*. 2018. Disponível em: <<https://github.com/ctic-sje-ifsc>>. Acesso em: 17 de novembro de 2018. Citado na página 29.
- Departamento de Infraestrutura e Serviços de Tecnologia da Informação. *Boas práticas, orientações e vedações para contratação de Serviços de Computação em Nuvem*. [S.l.], 2016. Disponível em: <<https://www.governodigital.gov.br/documentos-e-arquivos/Orientacao%20servicos%20em%20nuvem.pdf>>. Acesso em: 12 de maio 2018. Citado 2 vezes nas páginas 28 e 30.
- DIEZ, O.; SILVA, A. Govcloud: Using cloud computing in public organizations. *IEEE Technology and Society Magazine*, v. 32, n. 1, p. 66–72, 2013. ISSN 0278-0097. Citado na página 28.

DILLON, T.; WU, C.; CHANG, E. Cloud computing: Issues and challenges. In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. [S.l.: s.n.], 2010. p. 27–33. ISSN 1550-445X. Citado na página 31.

Docker Inc. *About Docker*. 2018. Disponível em: <<https://www.docker.com/company>>. Acesso em: 03 de junho de 2018. Citado na página 37.

Docker Inc. *What is a container*. 2018. Disponível em: <<https://www.docker.com/what-container>>. Acesso em: 02 de junho de 2018. Citado na página 37.

GREGOL, R. E. W. *Recursos de escalabilidade e alta disponibilidade para aplicações web*. 2011. Disponível em: <<http://repositorio.roca.utfpr.edu.br/jspui/handle/1/608>>. Acesso em: 30 de maio de 2018. Citado na página 36.

GROSMANN, D. et al. Estudo comparativo sobre o uso do vmware e xen server na virtualização de servidores. *ERCEMAPI*. Disponível em:< http://www.die.ufpi.br/ercemapi2011/artigos/ST3_17.pdf>. Acesso em, v. 1, n. 08, p. 2013, 2011. Citado na página 35.

GUDU, D.; HARDT, M.; STREIT, A. Evaluating the performance and scalability of the ceph distributed storage system. In: *2014 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2014. p. 177–182. Citado na página 43.

GÖRANSSON, P.; BLACK, C. *Software Defined Networks: A comprehensive approach*. Waltham, Estados Unidos: Morgan Kaufmann, 2014. Citado 2 vezes nas páginas 29 e 32.

HELM. *Rook: Design*. 2018. Disponível em: <<https://rook.github.io/docs/rook/master/>>. Acesso em: 21 de outubro de 2018. Citado na página 41.

Inktank Storage, Inc. *RADOS GATEWAY*. 2012. Disponível em: <<http://docs.ceph.com/docs/bobtail/radosgw/>>. Acesso em: 21 de outubro de 2018. Citado na página 44.

Inktank Storage, Inc. *ARCHITECTURE*. 2014. Disponível em: <<http://docs.ceph.com/docs/giant/architecture/>>. Acesso em: 21 de outubro de 2018. Citado na página 44.

IZRAILEVSKY, Y. *Completing the Netflix Cloud Migration*. 2016. Disponível em: <<https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration>>. Acesso em: 12 de maio 2018. Citado na página 28.

JOY, A. M. Performance comparison between linux containers and virtual machines. In: *2015 International Conference on Advances in Computer Engineering and Applications*. [S.l.: s.n.], 2015. p. 342–346. Citado 2 vezes nas páginas 37 e 38.

KAVIS, M. J. *Architecting The Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS and Iaas)*. New Jersey, Estados Unidos: John Wiley & Sons, 2014. Citado na página 27.

KHAN, A. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, v. 4, n. 5, p. 42–48, 2017. ISSN 2325-6095. Citado na página 39.

MEDEL, V. et al. Modelling Performance & Resource Management in Kubernetes. In: *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC 2016)*. IEEE, 2016. v. 9. Disponível em: <<https://books.google.com.br/books?id=gKnNAQAAcAAJ>>. Acesso em: 21 de maio 2018. Citado na página 29.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*: Recommendations of the national institute of standards and technology. [S.l.], 2011. Disponível em: <<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>>. Citado 4 vezes nas páginas 27, 31, 32 e 33.

MEYER, S.; MORRISON, J. P. Supporting heterogeneous pools in a single ceph storage cluster. In: *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. [S.l.: s.n.], 2015. p. 352–359. Citado na página 43.

Microsoft Azure. *Blob Storage*. 2018. Disponível em: <<https://azure.microsoft.com/en-us/services/storage/blobs/>>. Acesso em: 24 de novembro de 2018. Citado na página 29.

NETAPP. *What Is Converged Infrastructure (CI)?* 2018. Disponível em: <<https://www.netapp.com/us/info/what-is-converged-infrastructure.aspx>>. Acesso em: 24 de novembro de 2018. Citado na página 45.

NETO, M. C. et al. Authentication and availability in a flexible and adaptable distributed file system. In: *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 42.

Openstack. *Ceph RADOS Block Device (RBD)*. 2014. Disponível em: <<https://docs.openstack.org/juno/config-reference/content/ceph-rados.html>>. Acesso em: 21 de outubro de 2018. Citado na página 44.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, v. 2, n. 3, p. 24–31, May 2015. ISSN 2325-6095. Citado 2 vezes nas páginas 37 e 38.

PEARCE, M.; ZEADALLY, S.; HUNT, R. Virtualization: Issues, security threats, and solutions. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 45, n. 2, p. 17:1–17:39, mar. 2013. ISSN 0360-0300. Disponível em: <<http://doi.acm.org/10.1145/2431211.2431216>>. Citado na página 35.

PLESHAKOV, M. *Support for TCP/UDP Load Balancing*. 2018. Disponível em: <<https://github.com/nginxinc/kubernetes-ingress/tree/master/examples/tcp-udp>>. Acesso em: 03 de dezembro de 2018. Citado na página 41.

RANCHER LABS, INC. *What Rancher Adds To Kubernetes*. 2018. Disponível em: <<https://rancher.com/what-is-rancher/overview/>>. Acesso em: 21 de outubro de 2018. Citado na página 42.

Red Hat, Inc. *Ceph Documentation*. 2016. Disponível em: <<http://docs.ceph.com/docs/master/>>. Acesso em: 24 de novembro de 2018. Citado na página 43.

Red Hat, Inc. *INTRO TO CEPH*. 2016. Disponível em: <<http://docs.ceph.com/docs/mimic/start/intro/>>. Acesso em: 21 de outubro de 2018. Citado 2 vezes nas páginas 43 e 44.

Red Hat, Inc. *File storage, block storage, or object storage?* 2018. Disponível em: <<https://www.redhat.com/en/topics/data-storage/file-block-object-storage>>. Acesso em: 21 de outubro de 2018. Citado na página 43.

Rede Nacional de Pesquisa. *Programa Centros de Dados Compartilhados é inaugurado em Recife (PE)*. 2014. Disponível em: <<https://www.rnp.br/destaques/programa-centros-dados-compartilhados-e-inaugurado-recife-pe>>. Acesso em: 26 de maio de 2018. Citado na página 28.

ROOK. *Rook Documentation*. 2018. Disponível em: <<https://rook.github.io/docs/rook/master/>>. Acesso em: 24 de novembro de 2018. Citado na página 44.

ROOK. *The Kubernetes Helm Architecture*. 2018. Disponível em: <<https://docs.helm.sh/architecture/>>. Acesso em: 22 de outubro de 2018. Citado na página 52.

SADASHIV, N.; KUMAR, S. M. D. Cluster, grid and cloud computing: A detailed comparison. In: *2011 6th International Conference on Computer Science Education (ICCSE)*. [S.l.: s.n.], 2011. p. 477–482. Citado na página 34.

SINGH, S.; SINGH, N. Containers docker: Emerging roles future of cloud technology. In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. [S.l.: s.n.], 2016. p. 804–807. Citado 2 vezes nas páginas 37 e 38.

STEINBERGER, S. *Mountaineer, Forward*. 2017. Disponível em: <<https://pixabay.com/en/mountaineer-forward-blizzard-stormy-2080138/>>. Acesso em: 04 de novembro de 2018. Citado na página 58.

TANENBAUM, A. S. *Sistemas operacionais modernos*. 3. ed. [S.l.]: Prentice-Hall do Brasil, 2010. ISBN 9788576052371. Citado 2 vezes nas páginas 34 e 42.

TANENBAUM, A. S.; STEEN, M. V. *Sistemas Distribuídos*: princípios e paradigmas. 2. ed. São Paulo, Brasil: Pearson Education do Brasil, 1994. Citado 3 vezes nas páginas 29, 33 e 34.

TAY, Y. C.; GAURAV, K.; KARKUN, P. A performance comparison of containers and virtual machines in workload migration context. In: *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. [S.l.: s.n.], 2017. p. 61–66. Citado na página 35.

The Kubernetes Authors. *Ingress*. 2018. Disponível em: <<https://kubernetes.io/docs/concepts/services-networking/ingress/>>. Acesso em: 21 de outubro de 2018. Citado na página 41.

The Kubernetes Authors. *kubelet*. 2018. Disponível em: <<https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>>. Acesso em: 21 de outubro de 2018. Citado na página 39.

The Kubernetes Authors. *Pods*. 2018. Disponível em: <<https://kubernetes.io/docs/concepts/workloads/pods/pod/>>. Acesso em: 21 de outubro de 2018. Citado na página 40.

The Kubernetes Authors. *Services*. 2018. Disponível em: <<https://kubernetes.io/docs/concepts/services-networking/service/>>. Acesso em: 21 de outubro de 2018. Citado na página 40.

The Kubernetes Authors. *Storage Classes*. 2018. Disponível em: <<https://kubernetes.io/docs/concepts/storage/storage-classes/>>. Acesso em: 21 de outubro de 2018. Citado na página 40.

The Kubernetes Authors. *Understanding Kubernetes Objects*. 2018. Disponível em: <<https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>>. Acesso em: 21 de outubro de 2018. Citado na página 39.

The Kubernetes Authors. *Using kubectl to Create a Deployment*. 2018. Disponível em: <<https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/>>. Acesso em: 21 de outubro de 2018. Citado na página 40.

The Kubernetes Authors. *Using Minikube to Create a Cluster*. 2018. Disponível em: <<https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>>. Acesso em: 21 de outubro de 2018. Citado 2 vezes nas páginas 39 e 40.

The Kubernetes Authors. *What is Kubernetes?* 2018. Disponível em: <<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>>. Acesso em: 21 de outubro de 2018. Citado na página 39.

VMWARE. *Infraestrutura hiperconvergente (HCI)*. 2018. Disponível em: <<https://www.vmware.com/br/products/hyper-converged-infrastructure.html>>. Acesso em: 24 de novembro de 2018. Citado na página 45.

WALTERS, J. P. et al. A comparison of virtualization technologies for hpc. In: *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*. [S.l.: s.n.], 2008. p. 861–868. ISSN 1550-445X. Citado na página 35.

WATTS, J. *TOC Votes to Move Rook into CNCF Incubator*. 2018. Disponível em: <<https://www.cncf.io/blog/2018/09/25/toct-votes-to-move-rook/>>. Acesso em: 24 de novembro de 2018. Citado 2 vezes nas páginas 43 e 44.

WATTS, S. *SaaS vs PaaS vs IaaS: What's The Difference and How To Choose*. 2017. Disponível em: <<https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>>. Acesso em: 23 de junho de 2018. Citado na página 33.

Apêndices

APÊNDICE A – *SCRIPT DE INICIALIZAÇÃO DAS MÁQUINAS VIRTUAIS*

```
#!/bin/bash

# Install docker-ce 17.03.3
if [ ! -f /usr/bin/docker ]; then
    apt update
    wget -O docker.deb https://download.docker.com/linux/debian/dists/stretch/pool/
        stable/amd64/docker-ce_17.03.3~ce-0~debian-stretch_amd64.deb
    dpkg -i docker.deb
    apt -f -y install
    rm -rf docker*
fi

# Delete dataDir for rook monitors
rm -rf /var/lib/rook/*

# Enable rdb module for ceph
echo "rbd" > /etc/modules
```


APÊNDICE B – ARQUIVO DE CONFIGURAÇÃO DO TERRAFORM

```

variable "gce_ssh_user" {
  default = "matuzalemmuller"
}

variable "gce_ssh_pub_key_file" {
  default = "keys/id_rsa.pub"
}

data "template_file" "startup_script" {
  template = "${file("startup_script.sh")}"
}

provider "google" {
  credentials = "${file("keys/gcp.json")}"
  project     = "k8s-test-cluster-215603"
  region      = "southamerica-east1"
}

resource "google_compute_network" "network-0" {
  name = "network-0"
}

resource "google_compute_subnetwork" "subnet-0" {
  name         = "subnet-0"
  ip_cidr_range = "10.0.0.0/24"
  network      = "${google_compute_network.network-0.self_link}"
}

resource "google_compute_address" "address-0" {
  name         = "address-0"
  subnetwork   = "${google_compute_subnetwork.subnet-0.self_link}"
  address_type = "INTERNAL"
  address      = "10.0.0.100"
}

resource "google_compute_address" "address-1" {
  name         = "address-1"
  subnetwork   = "${google_compute_subnetwork.subnet-0.self_link}"
  address_type = "INTERNAL"
  address      = "10.0.0.101"
}

```

```
}

resource "google_compute_address" "address-2" {
  name        = "address-2"
  subnetwork  = "${google_compute_subnetwork.subnet-0.self_link}"
  address_type = "INTERNAL"
  address      = "10.0.0.102"
}

resource "google_compute_firewall" "firewall-0" {
  name      = "firewall-0"
  network   = "${google_compute_network.network-0.name}"

  allow {
    protocol = "icmp"
  }

  allow {
    protocol = "tcp"
    ports    = ["22", "53", "80", "443", "2380", "2379", "3306", "6443", "6790", "6800-7300", "8124", "9283", "10250", "10254", "30000-32767", "44134"]
  }

  allow {
    protocol = "udp"
    ports    = ["53", "8472"]
  }
}

resource "google_compute_instance" "vm-0" {
  name        = "vm-0"
  machine_type = "n1-standard-2"
  zone        = "southamerica-east1-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
      size  = "15"
    }
  }

  network_interface {
    subnetwork  = "${google_compute_subnetwork.subnet-0.name}"
    address     = "${google_compute_address.address-0.address}"
    access_config = {}
  }
}
```

```

metadata {
  sshKeys = "${var.gce_ssh_user}:${file(var.gce_ssh_pub_key_file)}"
}

metadata_startup_script = "${data.template_file.startup_script.rendered}"

tags = ["k8s"]
}

resource "google_compute_instance" "vm-1" {
  name        = "vm-1"
  machine_type = "n1-standard-2"
  zone        = "southamerica-east1-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
      size  = "15"
    }
  }
}

network_interface {
  subnetwork     = "${google_compute_subnetwork.subnet-0.name}"
  address       = "${google_compute_address.address-1.address}"
  access_config = {}
}

metadata {
  sshKeys = "${var.gce_ssh_user}:${file(var.gce_ssh_pub_key_file)}"
}

metadata_startup_script = "${data.template_file.startup_script.rendered}"

tags = ["k8s"]
}

resource "google_compute_instance" "vm-2" {
  name        = "vm-2"
  machine_type = "n1-standard-2"
  zone        = "southamerica-east1-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
      size  = "15"
    }
  }
}

```

```
network_interface {  
    subnetwork      = "${google_compute_subnetwork.subnet-0.name}"  
    address        = "${google_compute_address.address-2.address}"  
    access_config  = {}  
}  
  
metadata {  
    sshKeys = "${var.gce_ssh_user}:${file(var.gce_ssh_pub_key_file)}"  
}  
  
metadata_startup_script = "${data.template_file.startup_script.rendered}"  
  
tags = ["k8s"]  
}  
  
output "external-ip-0" {  
    value = "${google_compute_instance.vm-0.network_interface.0.access_config.0.  
        assigned_nat_ip}"  
}  
  
output "external-ip-1" {  
    value = "${google_compute_instance.vm-1.network_interface.0.access_config.0.  
        assigned_nat_ip}"  
}  
  
output "external-ip-2" {  
    value = "${google_compute_instance.vm-2.network_interface.0.access_config.0.  
        assigned_nat_ip}"  
}
```

APÊNDICE C – ARQUIVO DE CONFIGURAÇÃO DO RANCHER

```
nodes:
- address: 35.198.47.124
  internal_address: 10.0.0.100
  user: matuzalemmuller
  role: [controlplane,worker,etcd]
  ssh_key_path: keys/id_rsa
  hostname_override: master
  labels:
    node: master
- address: 35.198.60.116
  internal_address: 10.0.0.101
  user: matuzalemmuller
  role: [worker,etcd]
  hostname_override: worker1
  ssh_key_path: keys/id_rsa
  labels:
    node_role: worker
- address: 35.198.15.53
  internal_address: 10.0.0.102
  user: matuzalemmuller
  role: [worker,etcd]
  ssh_key_path: keys/id_rsa
  hostname_override: worker2
  labels:
    node_role: worker

kubelet:
  extra_args:
    feature-gates: MountPropagation=true
    volume-plugin-dir: /var/lib/kubelet/volumeplugins
  extra_binds:
    - "/var/lib/rook:/var/lib/rook:rshared"
```


APÊNDICE D – ARQUIVO DE CRIAÇÃO DO CLUSTER ROOK

```
# https://github.com/rook/rook/blob/master/cluster/examples/kubernetes/ceph/cluster
.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: rook-ceph
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rook-ceph-cluster
  namespace: rook-ceph
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-cluster
  namespace: rook-ceph
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: [ "get", "list", "watch", "create", "update", "delete" ]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-cluster-mgmt
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-cluster-mgmt
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph-system
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
```

```
metadata:
  name: rook-ceph-cluster
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rook-ceph-cluster
subjects:
- kind: ServiceAccount
  name: rook-ceph-cluster
  namespace: rook-ceph
---
apiVersion: ceph.rook.io/v1beta1
kind: Cluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  dataDirHostPath: /var/lib/rook
  serviceAccount: rook-ceph-cluster
  mon:
    count: 3
    allowMultiplePerNode: true
  dashboard:
    enabled: true
  network:
    hostNetwork: false
  resources:
    storage: # cluster level storage configuration and selection
      useAllNodes: true
      useAllDevices: false
      deviceFilter:
        location:
        config:
          databaseSizeMB: "1024"
          journalSizeMB: "1024"
```

APÊNDICE E – ARQUIVO DE INICIALIZAÇÃO DO *STORAGECLASS* ROOK

```
apiVersion: ceph.rook.io/v1beta1
kind: Pool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  replicated:
    size: 3
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
provisioner: ceph.rook.io/block
parameters:
  pool: replicapool
  clusterNamespace: rook-ceph
  fstype: ext4
```


APÊNDICE F – ARQUIVO INGRESS DA *OBJECT STORE*

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: rgw-object-store
  namespace: rook-ceph
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: nginx
spec:
  tls:
  - secretName: tls-secret
  rules:
  - host: www.wordpress.teia.cc
    http:
      paths:
      - path: /rook
        backend:
          serviceName: rook-ceph-rgw-my-store
          servicePort: 80
```


Anexos

ANEXO A – INSTRUÇÕES DE CRIAÇÃO DO CLUSTER KUBERNETES

Arquivo README.md contendo instruções de criação da infraestrutura remota e cluster *Kubernetes*. Disponível em <https://goo.gl/aZ3Grb>

README.md

This documentation presents step by step instructions on how to run a WordPress application with MySQL database in a remote Kubernetes cluster using Rook as the storage orchestrator.

- Below are the versions of each software used in this project:
 - Rook chart v0.8.2 (beta)
 - WordPress chart v2.1.10 (App v4.9.8 - stable)
 - MySQL chart v0.10.1 (App 5.7.14 - stable)
 - Docker v17.03.3
 - kubectl v1.11.0
 - Kubernetes v.v1.11.1

Table of contents

- [Set up remote infrastructure](#)
- [Install Rook Operator chart using Helm](#)
- [Create Rook cluster](#)
- [Install NGINX Controller](#)
- [Create Rook Storage Class](#)
- [Generate certificate for remote VMs](#)
- [Create secrets for Rook Object Store](#)
- [Create Rook Object Store](#)
- [Run Rook Toolbox](#)
- [Create S3 bucket using radosgw](#)
- [Create Ingress record for S3 bucket](#)
- [Create TLS secrets for WordPress](#)
- [Install MySQL chart](#)
- [Install WordPress chart](#)
- [Common issues](#)

Set up remote infrastructure

See [this documentation](#) for instructions on how to set up the remote infrastructure and local environment.

Install Rook

Install Rook Operator chart using Helm

Add the rook beta channel to Helm:

```
helm repo add rook-beta https://charts.rook.io/beta
```

Install the rook chart:

```
helm install rook-beta/rook-ceph --namespace rook-ceph-system --name rook-chart --set agent.flexVolumeDirPath=/v
```

Start the gcloud environment and connect to your account and project:

```
gcloud init
```

Modify `terraform-infrastructure.tf` file to include correct account information and credentials

Generate local SSH keys, which will be used to connect to the remote VMs. Save both keys (public and private) with default name (`id_rsa`) and place both keys inside the directory `remote-setup/keys`:

```
ssh-keygen -t rsa -b 4096 -C "email@domain.com"
```

Modify the file `remote-setup/terraform-infrastructure.tf` to include the user that will be created in the remote VMs and point to the project created in GCP (note that you should include the project ID):

```
(line 3) default = "user"  
...  
(line 18) project = "test-project-1234"
```

Download the JSON credential of the service account which will be used by Terraform to manage the GCP resources and save this file with the name "gcp.json" inside the "keys" directory: <https://console.cloud.google.com/apis/credentials/serviceaccountkey>.

Run Terraform and create the GCP infrastructure

Run the following commands within the `remote-setup` folder to see the changes that will be made by terraform, apply these changes and destroy them, respectively:

```
terraform plan  
terraform apply
```

The following command can be used to destroy the changes:

```
terraform destroy
```

Install docker in all VMs created

<https://docs.docker.com/install/linux/docker-ce/debian/#install-from-a-package>
<https://download.docker.com/linux/debian/dists/stretch/pool/stable/amd64/>

Give permissions to the user created in the remote VMs to run docker:

```
sudo usermod -aG docker $USER
```

Install rke in local computer

<https://rancher.com/docs/rke/v0.1.x/en/installation/>

Deploy remote k8s cluster

Modify the file `remote-setup/cluster.yml` to include the correct IPs and usernames so Rancher can access the VMs and create the cluster. After changing the file, deploy the remote k8s cluster using rke. The following command should be run within the `remote-setup` folder:

```
rke up --config ./cluster.yml
```

- Note that when the VMs were started using terraform all the necessary firewall rules should have been setup already, but you may also need to change your settings to allow additional ports.
- The docker version installed in the VM needs to be compatible with the rke version installed in the local computer. For example, at the time this project is being worked on (Q3 of 2018) the latest stable release of rke is 1.9.0, which supports docker-ce 17.03.x. However, this is not the latest version of Docker at this time.

Move k8s local file created by rancher to k8s local configuration folder

Move k8s configuration file created by rancher to the local configuration folder so k8s can locate the file and reach the nodes. Alternatively, you can also set the `KUBECONFIG` environmental variable to the path of `kube_config_cluster.yml`.

```
mv kube_config_cluster.yml config/ && cp config/kube_config_cluster.yml ~/.kube/config  
- or -  
export KUBECONFIG=$(pwd)/kube_config_cluster.yml
```

Install Helm in remote k8s cluster

See <https://github.com/helm/helm> for instructions on how to install Helm in your local computer.

Since RBAC is enabled in the cluster it's necessary to create a ServiceAccount and ClusterRoleBinding for the tiller service to manage charts.

```
kubectl --namespace kube-system create sa tiller
kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller
helm init
kubectl --namespace kube-system patch deploy/tiller-deploy -p '{"spec": {"template": {"spec": {"serviceAccountName": "tiller"}}}}'
```


ANEXO B – INSTRUÇÕES DE IMPLANTAÇÃO DO CLUSTER ROOK

Arquivo README.md contendo instruções de implantação do *cluster* Rook e aplicações MySQL e WordPress. Disponível em <https://goo.gl/Qe1PvX>

README.md

This documentation presents step by step instructions on how to run a WordPress application with MySQL database in a remote Kubernetes cluster using Rook as the storage orchestrator.

- Below are the versions of each software used in this project:
 - Rook chart v0.8.2 (beta)
 - WordPress chart v2.1.10 (App v4.9.8 - stable)
 - MySQL chart v0.10.1 (App 5.7.14 - stable)
 - Docker v17.03.3
 - kubectl v1.11.0
 - Kubernetes v1.11.1

Table of contents

- [Set up remote infrastructure](#)
- [Install Rook Operator chart using Helm](#)
- [Create Rook cluster](#)
- [Install NGINX Controller](#)
- [Create Rook Storage Class](#)
- [Generate certificate for remote VMs](#)
- [Create secrets for Rook Object Store](#)
- [Create Rook Object Store](#)
- [Run Rook Toolbox](#)
- [Create S3 bucket using radosgw](#)
- [Create Ingress record for S3 bucket](#)
- [Create TLS secrets for WordPress](#)
- [Install MySQL chart](#)
- [Install WordPress chart](#)
- [Common issues](#)

Set up remote infrastructure

See [this documentation](#) for instructions on how to set up the remote infrastructure and local environment.

Install Rook

Install Rook Operator chart using Helm

Add the rook beta channel to Helm:

```
helm repo add rook-beta https://charts.rook.io/beta
```

Install the rook chart:

```
helm install rook-beta/rook-ceph --namespace rook-ceph-system --name rook-chart --set agent.flexVolumeDirPath=/v
```

Installing the operator will create 7 pods:

- 3 rook agents, which will be running in each node
- 3 rook discover, which will be running in each node
- 1 rook operator, which will be running in the master node

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

matuzalem-macbook:rook matuzalem\$ kubectl get pod -n rook-ceph-system				
NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-agent-4vbxm	1/1	Running	0	27m
rook-ceph-agent-nzgtj	1/1	Running	0	27m
rook-ceph-agent-shxvj	1/1	Running	0	27m
rook-ceph-operator-67bf669467-cwzh6	1/1	Running	0	27m
rook-discover-2dhls	1/1	Running	0	27m
rook-discover-8g2rm	1/1	Running	0	27m
rook-discover-h2n8h	1/1	Running	0	27m

matuzalem-macbook:rook matuzalem\$ █

For more information about the configuration "agent.flexVolumeDirPath=/var/lib/kubelet/volumeplugins", visit [this link](#)

Create Rook Cluster

This will deploy a rook cluster with monitors (MON), OSDs and a manager (MGR). All the necessary requirements such as namespaces and roles will also be created. However, it will still be necessary to setup for what rook will be used (i.e. object store, filesystem, etc).

```
kubectl create -f k8s-deployment/rook/cluster.yaml
```

This command will create 10 pods:

- 3 monitors, which will be running in each node
- 3 osd prepare, which will run and complete in each node
- 3 osds, which will be running in each node
- 1 rook manager, which will be running in the master node

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

matuzalem-macbook:rook matuzalem\$ kubectl get pod -n rook-ceph				
NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-mgr-a-5b49f47b87-fzsjm	1/1	Running	0	24m
rook-ceph-mon0-66sjw	1/1	Running	0	25m
rook-ceph-mon1-k9mj9	1/1	Running	0	25m
rook-ceph-mon2-2xjwl	1/1	Running	0	25m
rook-ceph-osd-id-0-78674d5795-nvdxf	1/1	Running	0	24m
rook-ceph-osd-id-1-7bc6dcc877-8w2gs	1/1	Running	0	24m
rook-ceph-osd-id-2-5cc58d89b4-427gt	1/1	Running	0	24m
rook-ceph-osd-prepare-master-bb2l8	0/1	Completed	0	24m
rook-ceph-osd-prepare-worker1-48c8p	0/1	Completed	0	24m
rook-ceph-osd-prepare-worker2-dqfvrl	0/1	Completed	0	24m

Install WordPress chart and create Rook volume & bucket to store files

Install NGINX Controller

Install controller so http requests are forwarded to WordPress pod and Rook Object Store:

```
helm install stable/nginx-ingress --name nginx --set rbac.create=true
```

Create Rook Storage Class

Deploy Rook Storage Class. Volumes will now be created using rook:

```
kubectl create -f k8s-deployment/rook/storage-class.yaml
```

Generate certificate for remote VMs

- Point your domain to both worker VM IPs
- Generate a certificate using Let's Encrypt: <https://certbot.eff.org/lets-encrypt/debianstretch-other>
- Combine both `cert.pem` and `privkey.pem` in one file and encode output to base64. Insert the encoded output to the `cert` parameter of the `secrets.yaml` file

```
cat file.txt | base64
```

- Encode `privkey.pem` and `cert.pem` to base64 and add both to `k8s-deployment/rook/secrets.yaml` file in the `tls.key` and `tls.crt` parameters, respectively

Create secrets for Rook Object Store

Create TLS secrets for Rook Object Store and Object Store Ingress resource. This will allow secure connections to be established with the Object Store:

```
kubectl create -f k8s-deployment/rook/secrets.yaml
```

Create Rook Object Store

Create the Object Store, which will expose a S3 API to store and manage data:

```
kubectl create -f k8s-deployment/rook/object-store.yaml
```

- For more information about Rook Object Store, see <https://rook.io/docs/rook/master/object.html>
- A new pod will be created in namespace `rook-ceph`. Wait for its status to change to Running before proceeding to the next step

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-mgr-a-5b49f47b87-fzsjm	1/1	Running	0	1d
rook-ceph-mon0-66sjw	1/1	Running	0	1d
rook-ceph-mon1-k9mj9	1/1	Running	0	1d
rook-ceph-mon2-2xjwl	1/1	Running	0	1d
rook-ceph-osd-id-0-78674d5795-nvdxsf	1/1	Running	0	1d
rook-ceph-osd-id-1-7bc6dcc877-8w2gs	1/1	Running	0	1d
rook-ceph-osd-id-2-5cc58d89b4-427gt	1/1	Running	0	1d
rook-ceph-osd-prepare-master-bb2l8	0/1	Completed	0	1d
rook-ceph-osd-prepare-worker1-48c8p	0/1	Completed	0	1d
rook-ceph-osd-prepare-worker2-dqfvf	0/1	Completed	0	1d
rook-ceph-rgw-my-store-7878f8b699-9krgl	1/1	Running	0	1d

Run Rook Toolbox

Rook toolbox allows to connect to the cluster via CLI and analyze the underlying Ceph system running cluster, which helps troubleshooting issues. It will also allow to launch a S3 client to create buckets and manage data in the Rook Object Store.

```
kubectl create -f k8s-deployment/rook/toolbox.yaml
```

Create S3 bucket using radosgw

Access the rook toolbox pod and install the s3cmd client to manage data in the Rook Object Store (you can also simply deploy a `s3cmd` container such as [this one](#)):

```
kubectl -n rook-ceph exec -it rook-tools-XXX bash  
yum --assumeyes install s3cmd
```

Create rgw user to be able to manage data in the Object Store (still in the toolbox pod):

```
radosgw-admin user create --uid rook-user --display-name "A rook rgw User" --rgw-realm=my-store --rgw-zonegroup=
```

- Save the following output from the `radosgw-admin` command:

```
{  
    "user": "rook-user",  
    "access_key": "XXXXXXXXXXXXXXXXXXXXXX",  
    "secret_key": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  
}
```

In the toolbox shell, export the following variables to use them when managing data with s3cmd (the values of `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are variables from the previous step):

```
export AWS_HOST=rook-ceph-rgw-my-store.rook-ceph  
export AWS_ENDPOINT=rook-ceph-rgw-my-store.rook-ceph.svc.cluster.local  
export AWS_ACCESS_KEY_ID=XXXXXXXXXXXXXXXXXXXXXX  
export AWS_SECRET_ACCESS_KEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Create a S3 bucket using s3cmd:

```
s3cmd mb --no-ssl --host=${AWS_HOST} --host-bucket= s3://rookbucket
```

Save some data to later add to the bucket. For example, a picture:

```
curl -o image.jpg https://cdn.pixabay.com/photo/2017/02/19/16/01/mountaineer-2080138_960_720.jpg
```

- This image has no copyright and has been retrieved from [this page](#)

"Put" the data in the bucket created and change its permissions to public access:

```
s3cmd put image.jpg --no-ssl --host=${AWS_HOST} --host-bucket= s3://rookbucket  
s3cmd setacl s3://rookbucket/image.jpg --acl-public --no-ssl --host=${AWS_HOST} --host-bucket=s3://rookbucket
```

The following command can be used to list the objects stored in the bucket:

```
s3cmd ls s3://rookbucket --no-ssl --host=${AWS_HOST} --host-bucket=s3://rookbucket
```

- More s3cmd commands are available at <https://s3tools.org/usage>
- The flag `--no-ssl` is being used as this communication is internal to the cluster

Create Ingress record for S3 bucket

Add the domain that was previously used to create the certificate and is pointing to the remote worker nodes (VMs) to the `host` parameter of the `k8s-deployment/rook/object-ingress.yaml` file:

```
(line 13) host: _____
```

Create Ingress record for S3 bucket:

```
kubectl create -f k8s-deployment/rook/object-ingress.yaml
```

- You will now be able to access your image from outside the cluster over HTTPS by accessing the URL www.domain.com/rook/rookbucket/image.jpg (where www.domain.com is the domain that was previously used to create the certificate and is pointing to the remote worker nodes - VMs).

Install MySQL chart

Run MySQL database which will be used by WordPress:

```
helm install stable/mysql --name mysql --version v0.10.1 -f k8s-deployment/wordpress/mysql-values.yaml
```

- This will install WordPress and create volumes based in the storage class `rook-ceph-block`
- More configurable parameters can be checked at <https://github.com/helm/charts/tree/master/stable/mysql>

Create TLS secrets for WordPress

Encode the `privkey.pem` and `cert.pem` files generated from the certificate to base64 and add both to `k8s-deployment/wordpress/wordpress-secrets.yaml` file in the `tls.key` and `tls.crt` parameters, respectively.

Create TLS secrets so it's possible to connect to WordPress securely:

```
kubectl create -f k8s-deployment/wordpress/wordpress-secrets.yaml
```

Install WordPress chart

Change the `host` parameter in the `k8s-deployment/wordpress/wordpress-values.yaml` file to include the domain that is pointing to the remote nodes:

```
(line 100) - name: _____
```

Install WordPress chart using Helm:

```
helm install stable/wordpress --name wordpress --version v2.1.10 -f k8s-deployment/wordpress/wordpress-values.yaml
```

- This will install WordPress and create volumes based in the storage class `rook-ceph-block`
- More configurable parameters can be checked at <https://github.com/helm/charts/tree/master/stable/wordpress>

Common issues

- Can't install chart because there's already a chart with that name installed even though it was removed: delete chart again using `--purge` flag

```
helm delete __chart__ --purge
```

- `rook-ceph` namespace stuck in terminating status: <https://github.com/rook/rook/issues/1488#issuecomment-397241621>

```
kubectl -n rook-ceph patch clusters.ceph.rook.io rook-ceph -p '{"metadata": {"finalizers": []}}' --type=merge
```

- `rook-ceph-system` namespace is still available after deleting chart: this is a known issue described [here](#). It is necessary to manually remove the resources created by the chart even after deleting the chart.
- Monitors failing to start: <https://github.com/rook/rook.github.io/blob/master/docs/rook/v0.7/common-problems.md#failing-mon-pod>
- OSDs failing to start: <https://github.com/rook/rook.github.io/blob/master/docs/rook/v0.7/common-problems.md#osd-pods-are-failing-to-start>
- Volume creation doesn't work: <https://github.com/rook/rook.github.io/blob/master/docs/rook/v0.7/common-problems.md#volume-creation>