MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

TECNOLOGIAS DE GERENCIAMENTO DE CONTÊINERES PARA APLICAÇÕES GEOESPACIAIS

João Pedro Diehl

Relatório de Iniciação Científica, orientada pelo Dr. Gilberto Ribeiro de Queiroz e Dr. Rennan Bezerra de Freitas Marujo

URL do documento original:
<http://urlib.net/>

INPE
São José dos Campos
2023

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

TECNOLOGIAS DE GERENCIAMENTO DE CONTÊINERES PARA APLICAÇÕES GEOESPACIAIS

João Pedro Diehl

Relatório de Iniciação Científica, orientada pelo Dr. Gilberto Ribeiro de Queiroz e Dr. Rennan Bezerra de Freitas Marujo

URL do documento original:
<http://urlib.net/>

INPE
São José dos Campos
2023

Dados Internacionais de Catalogação na Publicação (CIP)

Sobrenome, Nomes.

Cutter Tecnologias de Gerenciamento de Contêineres para Aplicações Geoespaciais / Nome Completo do Autor1; Nome Completo do Autor2. – São José dos Campos : INPE, 2023.

vii + 32 p. ; ()

Dissertação ou Tese (Mestrado ou Doutorado em Nome do Curso) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, AAAA.

Orientador : José da Silva.

1. Palavra chave. 2. Palavra chave 3. Palavra chave. 4. Palavra chave. 5. Palavra chave I. Título.

CDU 000.000

RESUMO

O uso de tecnologias de virtualização vem crescendo nos últimos anos. Em particular, as tecnologias de conteinerização têm sido amplamente utilizadas de maneira complementar às tecnologias de virtualização de hardware por oferecerem um ambiente de execução leve, com grande portabilidade e isolamento. Os contêineres permitem a execução de aplicativos em ambientes isolados e são capazes de encapsular os códigos das aplicações junto de suas bibliotecas e dependências. Os orquestradores de contêineres são um avanço importante no gerenciamento de aplicações em contêineres que, frequentemente, estão distribuídos em conjuntos de *clusters* de servidores. Essas tecnologias trazem vantagens como maior facilidade na escalabilidade das aplicações, maior eficiência na utilização de recursos através do balanceamento de cargas de trabalho e maior facilidade na distribuição e replicação das aplicações na infraestrutura computacional disponível. Todas essas qualidades são importantes para aplicações geoespaciais desenvolvidas no INPE e no projeto Brazil Data Cube (BDC), que lidam com o armazenamento e processamento de grandes volumes de dados, e, consequentemente, podem se beneficiar com uma maior automação da escalabilidade, distribuição e tolerância a falhas. Este projeto de iniciação científica tem por objetivo o estudo e implantação do serviço BDC-STAC, uma implementação desenvolvida pelo projeto Brazil Data Cube do padrão *SpatioTemporal Asset Catalog*, em um *cluster* de contêineres gerenciado por um orquestrador e implementado em máquinas virtuais disponibilizadas na infraestrutura computacional do BDC. Inicialmente, foi realizada uma revisão bibliográfica de tecnologias de conteinerização e utilizou-se o serviço BDC-STAC em um ambiente controlado. Para a implementação do *cluster*, foi escolhido o uso da plataforma de orquestração *Kubernetes* empregando o contêiner *runtime Docker*. Foram criados arquivos manifestos, que são conjuntos de instruções para a implantação de aplicações em *Kubernetes*, para o serviço BDC-STAC e para o banco de dados ao qual ele se conecta. Esses arquivos, bem como suas configurações, foram disponibilizados em um repositório público. O banco de dados foi inicializado em uma única réplica, enquanto o serviço BDC-STAC pôde ser inicializado em múltiplas réplicas, de modo que todas as réplicas utilizassem o mesmo banco de dados.Para além da variação no número de réplicas do *cluster* com o serviço BDC-STAC, ainda há possibilidade de melhoras no ambiente. Uma das possibilidade é avaliar outras arquiteturas, como por exemplo utilizar banco de dados com réplicas. Entretanto, essa abordagem aumenta a complexidade do ambiente, uma vez que é necessário que as instâncias dos servidores de bancos de dados mantenham-se sincronizadas e consistentes. Outros serviços da plataforma BDC, em especial os de visualização de dados, também podem se beneficiar de melhorias fornecidas por uma implantação em ambiente *Kubernetes*. Desta forma, uma vez implementadas e avaliadas diferentes arquiteturas do serviço BDC-STAC no ambiente *Kubernetes*, outros serviços do projeto BDC também podem adaptados para esse ambiente.

Palavras-chave: Brazil Data Cube. SpatioTemporal Asset Catalog. Geoespacial. Kubernetes. Orquestração de Contêineres.

LISTA DE FIGURAS

Pág.

SUMÁRIO

Pág.

1 INTRODUÇÃO

As tecnologias geoespaciais são fundamentais em diversas áreas pois elas possibilitam o gerenciamento de informações geográficas. O Sistema de Informação Geoespacial (GIS, do inglês *Geospatial Information System*) é uma tecnologia geoespacial que facilita o armazenamento, análise, gerenciamento e visualização de dados espaciais (SCHOLTEN et al., 2009). Dessa forma, áreas que lidam com o armazenamento e processamento de dados geoespaciais, como a área de observação da Terra (EO, do inglês *Earth Observation*), se beneficiam de tecnologias geoespaciais como o GIS.

Devido ao aumento da demanda de armazenamento e processamento de dados nos últimos anos, novas tecnologias foram desenvolvidas para a área de EO (FERREIRA et al., 2020). Dentre elas, o cubo de dados é uma nova solução para o armazenamento, gerenciamento e análise de dados de EO e, geralmente, contém as dimensões espaço e tempo (GIULIANI et al., 2017). Esse conceito foi implementado pelo INPE no projeto *Brazil Data Cube* (BDC) (FERREIRA et al., 2020). Para o BDC, uma tecnologia fundamental para o funcionamento e visualização de um cubo de dados é a catalogação de ativos espaço-temporais (ZAGLIA et al., 2019), que são qualquer dado sobre a Terra coletado em um determinado espaço e tempo. O *SpatioTemporal Asset Catalog* (STAC) (STAC, 2023) é uma especificação que define um padrão de metadados para catalogação de ativos geoespaciais. Essa especificação foi implementada pelo BDC para às necessidades do projeto, recebendo o nome BDC-STAC. O BDC-STAC fornece uma API web, para consulta, acesso e descoberta de diversas coleções de imagens de cubo dados, aos clientes do BDC. Atualmente, esse serviço é implementado em de modo a não ser executado em múltiplas instâncias.

Nas últimas décadas houve um avanço exponencial das tecnologias da informação e a demanda por recursos computacionais tem aumentado (SIDDIQUI et al., 2019). Por isso, novas tecnologias vêm sendo desenvolvidas buscando aumentar a eficiência no uso desses recursos. Dentre elas, as tecnologias de virtualização tem ganhado popularidade nos últimos anos. Essas tecnologias possibilitam que múltiplas instâncias de aplicações compartilhem recursos dinamicamente, o que torna cargas de trabalho mais escaláveis, econômicas e eficientes (SHARMA et al., 2016; RISTA et al., 2020). As duas principais tecnologias de virtualização são as máquinas virtuais (VMs) e os contêineres. A virtualização a nível de hardware é capaz de virtualizar os recursos físicos para múltiplas VMs, cada uma com um sistema operacional (SO) isolado das demais (SHARMA et al., 2016). Os contêineres são um mecanismo leve de virtualização a nível de SO e compartilham o *kernel* do sistema hospedeiro. Eles possibilitam

o encapsulamento dos códigos de aplicações junto de suas bibliotecas e dependências e oferecem, em um ambiente leve de execução, portabilidade e isolamento.

Devido às vantagens no uso de contêineres, a adoção dessa tecnologia vem crescendo. Segundo Jawarneh et al. (2019), a conteinerização é parte importante da arquitetura baseada em microsserviços que permite a decomposição de aplicações monolíticas em componentes independentes facilitando o gerenciamento e escalabilidade delas. Nessa arquitetura, aplicações podem ser formadas por centenas ou milhares de contêineres distribuídos em conjuntos de *clusters* de servidores. Essa característica dificulta o uso de contêineres em grandes aplicações, o que levou ao desenvolvimento de uma tecnologia voltada para o gerenciamento de contêineres, os orquestradores de contêineres. Esses sistemas têm o objetivo de gerenciar aplicações conteinerizadas e trazem diversas vantagens. Dentre elas, é possível citar: maior eficiência na utilização de recursos através do balanceamento de cargas de trabalho, maior facilidade de escalabilidade e maior tolerância a falhas (RODRIGUEZ; BUYYA, 2019). Atualmente, um dos orquestradores mais populares é o *Kubernetes*. Tendo em vista os benefícios de abordagens que implementam sistemas distribuídos, o serviço BDC-STAC pode se beneficiar de uma implementação conteinerizada gerenciada pelo *Kubernetes*.

Uma das vantagens da tecnologia de virtualização, em particular, da tecnologia de conteinerização, é fornecer uma nova abordagem para garantir a reprodutibilidade de pesquisas científicas. Apesar de todos os passos do processo científico dependerem progressivamente mais de computação e algoritmos, essa mudança não gerou maior garantia de reprodutibilidade. Processos científicos essenciais como replicar resultados ou testar a conclusão em outros contextos ficaram muito mais complexos (BOETTIGER, 2015). Tecnologias de conteinerização auxiliam na reprodutibilidade fornecendo um ambiente de testes pronto e portátil em que a aplicação e suas dependências estão instaladas.

O objetivo deste projeto de iniciação científica é o estudo e implantação do serviço BDC-STAC em um *cluster*, gerenciado por uma plataforma de orquestração de contêineres, implementado em máquinas virtuais disponibilizadas na infraestrutura computacional do BDC.

Este trabalho é organizado da seguinte forma. O Capítulo 2 apresenta uma revisão de conceitos importantes relacionados à tecnologias geoespaciais e de conteinerização. O Capítulo 3 apresenta a implantação do serviço BDC-STAC em um *cluster* Kubernetes. No Capítulo 4 são apresentados os arquivos manifestos utilizados para

a implantação do serviço BDC-STAC no *cluster* e os resultados dos testes de comparação das arquiteturas. Por fim, o Capítulo 5 fornece uma discussão sobre o estudo e os próximos passos do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Tecnologias geoespaciais

Os avanços tecnológicos dos séculos XX e XXI permitiram o desenvolvimento e a difusão de tecnologias da informação e comunicação (TICs). Essas tecnologias são capazes de armazenar, manipular, enviar e receber dados digitais (SCHOLTEN et al., 2009). Entre as TICs, as tecnologias geoespaciais são fundamentais em diversas áreas pois elas possibilitam o gerenciamento de informações geográficas. Atualmente, o uso dessas tecnologias se dá desde em projetos de captura e análise de dados geoespaciais por entidades governamentais e acadêmicas até em mapas presentes nos dispositivos móveis de bilhões de indivíduos. Devido ao extenso uso de tecnologias geoespaciais no mundo atual, foi criada a organização *Open Geospatial Consortium* (OGC) que é responsável pela padronização de serviços geográficos abertos (BAUMANN, 2010).

Um campo importante das tecnologias geoespaciais é o Sistemas de Informação Geográfica. Esse tipo de ferramenta foi desenvolvida com o objetivo de auxiliar no armazenamento, análise, gerenciamento e visualização de dados espaciais que relacionam elementos matriciais ou vetoriais a sistemas de coordenadas possibilitando a detecção de padrões espaciais (SCHOLTEN et al., 2009; BODENHAMER, 2012). Geralmente, esses sistemas também contam com um banco de dados geográfico que é voltado para o armazenamento de dados espaciais (FURTADO, 2018). Esses bancos de dados permitem a representação de objetos geométricos como pontos, linhas e polígonos.

De acordo com Shaw et al. (2008), o desenvolvimento dos GIS teve influência da cartografia e, como consequência, o foco dessas ferramentas esteve na representação de fenômenos geográficos estáticos, não oferecendo a representação da dimensão temporal. Nas últimas décadas houveram diversos avanços nos sistemas de informações geográficas, dentre eles, foram desenvolvidas novas funções para representação e análise de conjuntos de dados espaço-temporais. Um GIS espaço-temporal possibilita a análise, representação e visualização de dados nas dimensões do espaço e tempo e simplifica o estudo de tendências e padrões espaço-temporais.

Um uso importante do GIS espaço-temporal está na área de observação da Terra. Segundo Sudmanns et al. (2020), dados de observação da Terra são fundamentais no estudo de processos naturais e antropogênicos no sistema terrestre. Porém, devido a abundância no volume de dados coletados in situ e por sensoriamento remoto, analisar e extrair informações relevantes de padrões espaço-temporais relacionadas

a atividades humanas e processos naturais é um desafio. Devido ao grande volume de dados necessários em algumas análises, os fluxos de trabalhos através da Web para o computador do usuário estão se tornando inviáveis.

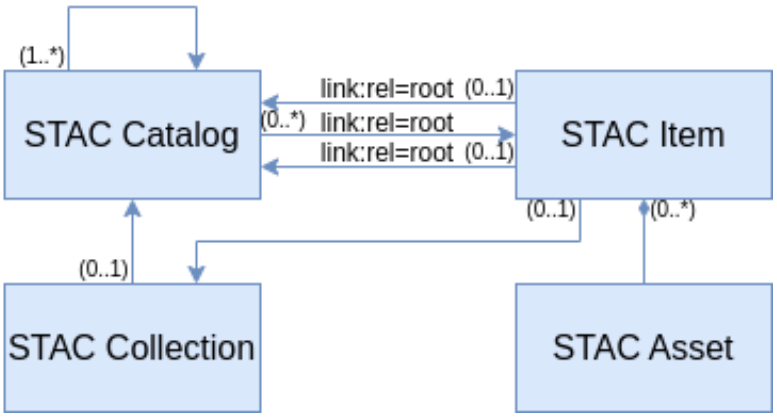
Nos últimos anos uma nova solução para armazenar, gerenciar e analisar dados de observação da Terra está sendo desenvolvida (GIULIANI et al., 2017). Chamado cubo de dados, esse conceito já foi integrado em projetos de múltiplos países, a exemplo do *Swiss Data Cube* (SDC) (GIULIANI et al., 2017) e do *Brazil Data Cube* (BDC) (FERREIRA et al., 2020). Um cubo de dados geralmente contêm as dimensões espacial e temporal, podendo representar uma série temporal de imagens de satélite (BAUMANN et al., 2018). Os dados podem ser ordenados ao longo do eixo temporal além dos eixos espaciais (SUDMANNS et al., 2020). Ainda não há um consenso sobre os conceitos de cubos de dados, podendo eles apresentarem, ou não, mesma resolução espacial e temporal entre seus itens, por exemplo.

Metadados são essenciais para a utilização de conjuntos de dados pois fornecem informações de como os dados devem ser consumidos ou interpretados (BAUMANN et al., 2018). Ao longo dos anos, foram desenvolvidas diversos padrões de catalogação que definem como os metadados de ativos geoespaciais devem ser estruturados e consultados. Dentre eles, o *SpatioTemporal Asset Catalog* (STAC)[1] que é um padrão que fornece uma estrutura comum para catalogar metadados de arquivos que representam informações sobre a Terra capturadas em um espaço e tempo.

A especificação STAC baseia-se em 4 principais entidades: *Catalog*, *Colection*, *Item* e *Asset*. A Figura 2.1 demonstra os relacionamentos entre tais entidades.

[1] https://stacspec.org/en

Figura 2.1 - Entidades da especificação STAC.



Fonte: Adaptado de Zaglia (2020)

As entidades do STAC definem tipos de objetos JSON relacionados e, usualmente, diversas especificações são combinadas para formar uma implementação. A estrutura mais comum para uma implementação é composta pelas especificações *Item*, *Catalog*, *Colection*, *Asset* e API. Essas especificações são hierárquicas e foram desenvolvidas para serem utilizadas em conjunto. Segundo STAC (2023):

- **STAC *Item*** é definido por uma *Feature* em *GeoJSON* e é uma coleção atômica de dados e metadados inseparáveis.
- **STAC *Catalog*** fornece uma estrutura flexível para conectar múltiplos STAC *Items* que podem ser navegados. A especificação contém uma lista de STAC *Items* e uma lista de STAC *Catalogs* filhos o que permite o agrupamento hierárquico de STAC *Items*.
- **STAC *Collection*** fornece informações adicionais sobre uma coleção de dados espaço-temporais. Possibilita a extensão do *Catalog* acrescentando campos para descrição de informações adicionais.
- **STAC *Asset***: um *asset* consiste em algum recurso (arquivo *raster*, arquivo vetorial, arquivo comprimido, arquivo de texto), contendo informações sobre um determinado STAC *item*.
- **STAC API** define uma interface de serviço *RESTful* para busca. É capaz de criar dinamicamente, como reposta a uma consulta de usuário, uma

FeatureCollection de STAC *Items* requisitados em formato *GeoJSON*.

Segundo (ZAGLIA et al., 2019), é possível implementar dois tipos de catálogos, o estático e o dinâmico. Os catálogos estáticos são arquivos JSON navegáveis através de links. Os catálogos dinâmicos geram suas repostas dinamicamente através de requisições a um serviço Web que adote a especificação STAC API.

No projeto *Brazil Data Cube* (BDC) foi desenvolvida uma implementação do STAC para a catalogação de metadados do BDC. O BDC-STAC é um catálogo dinâmico que implementa a STAC API e é utilizado para fornecer as coleções e cubos de dados aos usuários do BDC. As requisições são tratadas utilizando o micro *framework* Flask do Python e a partir delas, são realizadas consultas no banco de dados que contém os metadados do BDC (ZAGLIA et al., 2019).

2.2 Virtualização

Virtualização é uma abstração dos recursos computacionais (CHAE et al., 2019). Existem dois tipos de tecnologia de virtualização relevantes atualmente: a virtualização a nível de hardware e a virtualização a nível de sistema operacional.

A virtualização a nível de hardware utiliza uma aplicação chamada gerenciador de máquina virtual ou hiper-visor que é responsável por emular o hardware da máquina física para cada máquina virtual criada. Cada máquina virtual, então, consegue utilizar um sistema operacional próprio que é independente do sistema operacional da máquina física (SHARMA et al., 2016). O sistema da máquina virtual é chamado de convidado e o sistema da máquina física é chamado de hospedeiro. A estrutura de uma máquina virtual é, tipicamente, da seguinte forma: o sistema hospedeiro é instalado no hardware físico e é responsável por executar o hiper-visor que por sua vez, é responsável por executar a máquina virtual que é composta de um sistema operacional independente e aplicações e dependências (BARIK et al., 2016).

A virtualização a nível de sistema operacional utiliza ferramentas do *kernel* de um sistema operacional para que seja possível criar múltiplas instâncias isoladas de espaço de usuário. Estas instâncias são chamadas de contêineres (CHAE et al., 2019). Cada contêiner é um grupo de processos isolado de outros processos no sistema e, por isso, o *kernel* do sistema hospedeiro é compartilhado com os contêineres. São disponibilizados recursos do sistema para cada contêiner e o uso desses recursos pode ser limitado e monitorado (SHARMA et al., 2016). A estrutura de um contêiner é da seguinte forma: o sistema hospedeiro, que pode ser de uma máquina física ou

virtual, executa a aplicação responsável por gerenciar as imagens de contêiner. Essa aplicação pode ser um contêiner *runtime* ou contêiner *engine*. A imagem de contêiner tem o binário e as dependências necessárias para executar uma aplicação (BARIK et al., 2016).

O contêiner que executa apenas um processo ou aplicação é chamado de contêiner de aplicação e o contêiner que simula um sistema operacional completo é chamado de contêiner de sistema (BARIK et al., 2016). Por isso, contêineres de sistema costumam ser duradouros e hospedam várias aplicações ao passo que contêineres de aplicação geralmente lidam com cargas de trabalho que são efêmeras, ou seja, os contêineres são temporários e é possível criá-los e deletá-los com facilidade.

A arquitetura de um contêiner pode ser dividida em dois componentes principais, contêiner *runtime* e contêiner *engine*, o segundo sendo opcional para a implementação do contêiner. Contêiner *runtime* é o componente responsável por de fato gerenciar o ciclo de vida do contêiner e pode ser dividido em baixo nível e alto nível. Um contêiner *runtime* de baixo nível performa tarefas de baixo nível para executar um contêiner. Exemplos de suas responsabilidades são: se comunicar com o *kernel* para iniciar processos conteinerizados e configurar *cgroups* (MCCARTY, 2018). Um dos contêiner *runtimes* de baixo nível mais utilizados é o *runc*, uma implementação de referência do *runtime-spec* definido pela *Open Container Initiative* (OCI), uma iniciativa que visa projetar padrões abertos para virtualização em nível de sistema operacional. Um contêiner *runtime* de alto nível suporta ferramentas de alto nível como, por exemplo, recursos úteis para o gerenciamento de imagens de contêiner. Geralmente fornecem um *daemon* de aplicativo e uma Interface de Programação de Aplicação (API, do inglês *Application Programming Interface*) que aplicações remotas podem usar para monitorar os contêineres (MCCARTY, 2018). Exemplos de contêiner *runtimes* de alto nível são o *containerd*[2] e o CRI-O[3].

O *containerd* é um *daemon-based runtime* de alto nível que surgiu a partir do *Docker*[4] e é compatível com a *Container Runtime Interface* (CRI), uma interface que facilita a utilização de uma variedade de contêiner *runtimes* na plataforma de orquestração *Kubernetes*[5]. O *containerd* é responsável por gerenciar o ciclo de vida completo dos contêineres incluindo a execução e monitoramento deles. Implementa o download e gerenciamento de imagens e as entrega a um *runtime* de baixo nível (por padrão,

[2]https://containerd.io/
[3]https://cri-o.io/
[4]https://www.docker.com/
[5]https://kubernetes.io/

o *runC*) que, então, utiliza os recursos do *kernel* para execução de contêineres a partir das imagens, porém não implementa criação de imagem. Permite gerenciar ciclos de vida de *containers* através de uma API e aplicação cliente para interação. O CRI-O é um contêiner *runtime* que foi criado para permitir que o *Kubernetes* execute contêineres sem muito código ou ferramentas externas. É uma alternativa mais leve ao uso do *Docker* como *runtime* no *Kubernetes*.

Contêiner *engines* são plataformas que facilitam o gerenciamento e a criação de contêineres. Essas plataformas aceitam requisições do usuário e, da perspectiva dele, executam o contêiner (MCCARTY, 2018). Em geral, contêiner *engines* não são responsáveis por executar os contêineres e, portanto, dependem de contêiner *runtimes* para essa tarefa. O *Docker* é um exemplo de contêiner *engine*, sendo um dos pioneiros na tecnologia de contêiner de aplicação e ainda hoje, um dos mais populares.

2.3 *Kubernetes*

Plataformas de orquestração de contêineres emergiram com a popularização do uso de contêineres. Foram desenvolvidas para o gerenciamento de aplicações conteinerizadas em *clusters* de larga escala e são capazes de executar centena de milhares de trabalhos alocados em milhares de máquinas (RODRIGUEZ; BUYYA, 2019).

Uma das plataformas de orquestração de contêineres mais populares atualmente é *Kubernetes*, um orquestrador disponível em código aberto que foi desenvolvido para gerenciar, em *clusters*, cargas de trabalho conteinerizadas (RODRIGUEZ; BUYYA, 2019). *Kubernetes* simplifica a implantação, gerenciamento e execução de contêineres em sistemas distribuídos e é capaz de melhorar a eficiência na utilização de recursos, distribuição de cargas de trabalho e tolerância a falhas através de replicação de contêineres (ROSSI et al., 2020).

A arquitetura de um *cluster Kubernetes* pode ser dividida em: uma camada de gerenciamento composta por um ou mais nós chamados *master nodes* que executam processos essenciais para o funcionamento do *cluster* como, por exemplo, decidir em qual nó será alocado um determinado *pod*; e um conjunto de nós de processamento chamados *worker nodes* que executam as aplicações conteinerizadas (KUBERNETES, 2023).

A comunicação entre a camada de gerenciamento e o conjunto de nós de processamento é feita através de componente chamado de API *server* que expõe API do *Kubernetes*. A implementação principal desse componente é o *kube-apiserver*. Outro

componente importante da camada de gerenciamento é o *etcd*, um armazenamento de chave-valor de código aberto e distribuído. Ele é utilizado como armazenamento de apoio do *Kubernetes* para todos os dados do *cluster* (KUBERNETES, 2023).

O componente básico do conjunto de nós de processamento é o *pod*, uma abstração sobre os contêineres que permite que interações de usuários sejam com a camada do *Kubernetes* e que o *cluster* seja independente da tecnologia de contêiner (KUBERNETES, 2023). Em cada nó de um *cluster* podem haver múltiplos *pods* em execução que são gerenciados pelo *kubelet*, um processo presente em todos os nós para garantir que os *pods* estejam funcionando corretamente. Geralmente, para cada *pod*, há uma única instância de uma aplicação implantada em um contêiner ou um grupo pequeno de contêineres fortemente acoplados e, para cada contêiner, é possível configurar requisições e limites de recursos computacionais (ROSSI et al., 2020). Essas requisições são utilizados na criação dos *pods* e são especificadas em arquivos de descrição, chamados manifestos, escritos em formato *yaml*.

O *scheduler* é um componente da camada de gerenciamento que assegura que o total de recursos computacionais requisitados por todos os pods em um nó não ultrapasse a capacidade total do nó (RODRIGUEZ; BUYYA, 2019). Quando uma aplicação está para ser executada no *cluster*, o *master node* utiliza o *scheduler*, que tem como implementação padrão o *kube-scheduler*, para identificar os *worker* nodes mais apropriados para receberem os *pods* da aplicação (ROSSI et al., 2020).

O *pod* é um componente efêmero e pode ser criado e destruído dinamicamente. Cada vez que é criado, um *pod* recebe um endereço de IP único. Para que os *pods* sejam acessíveis a nós externos, é utilizado um método chamado *service* que expõe uma aplicação de rede que está sendo executada como um ou mais *pods*. Quando um *pod* é destruído, o *service* permanece, pois seus ciclos de vida não estão vinculados. (KUBERNETES, 2023)

O *service* e o *deployment* são dois objetos muito utilizados em *clusters Kubernetes*. O *service* é usado para comunicação entre os *pods* e o *Deployment* descreve o estado desejado para uma aplicação como, por exemplo, o número de réplicas. Para agrupar o *service* e o *deployment* são utilizados *labels* e *selectors*. (KUBERNETES, 2023)

Em cenários onde existem muitos grupos de componentes distintos em um *cluster*, é possível isolá-los utilizando a ferramenta *namespaces*. Essa ferramenta fornece um escopo para nomes, ou seja os nomes dos recursos não precisam ser diferentes entre *namespaces*, apenas dentro do mesmo. (KUBERNETES, 2023)

Para aplicações que dependem de armazenamento persistente, foram desenvolvidos os objetos *Persistent Volume* (PV) e *Persistent Volume Claim* (PVC). Os PVs são recursos do *cluster* usados no gerenciamento de armazenamentos persistentes. Eles têm o ciclo de vida independente dos *pods* e podem ser dinamicamente provisionados, ou seja, não são deletados junto aos *pods* e não é necessário que o usuário crie ou delete os PVs. PVCs são requisições por recursos PV. As requisições podem variar, por exemplo, serem de tamanhos específicos ou por modos de acesso. As PVCs são usadas pelos *pods* no lugar de volumes. (KUBERNETES, 2023)

O *Ingress* é um objeto que controla o acesso externo ao *cluster*. Ele expõe as rotas HTTP e HTTPS de fora *cluster* para o serviços internos ao *cluster*. Um *Ingress* pode agir como um balanceador de cargas e possibilitar que serviços sejam acessados por URLs externa. (KUBERNETES, 2023)

Desde o surgimento do *Kubernetes* foram desenvolvidas diversas distribuições e ferramentas para o gerenciamento de *clusters*. Uma distribuição popular é o K3s que é voltado para ambientes com restrição de recursos. Já, o *Rancher* é uma ferramenta que facilita o uso do *Kubernetes* fornecendo uma interface gráfica e uma *stack* de softwares que conta com, entre outros, o *Longhorn* que é um sistema de armazenamento persistente e distribuído para o *Kubernetes*.

3 MÉTODOS

Após a revisão bibliográfica de tecnologias geoespaciais e de virtualização, foi projetado um estudo visando implantar o serviço *BDC-STAC* em um *cluster Kubernetes*.

3.1 Implantação do BDC-STAC utilizando *Docker*

Na primeira etapa do estudo, seguindo a documentação do serviço BDC-STAC[1], foi possível realizar a implantação dele com o *Docker* a partir do *Dockerfile* disponibilizado no repositório público do BDC-STAC[2]. Para isso, o sistema operacional utilizado foi o *Ubuntu* 22.04 instalado em uma máquina física. O *Ubuntu* foi escolhido por ser um SO livre e de código aberto que utiliza, como *kernel*, o *Linux* e conta com grande suporte e disponibilidade de softwares.

Inicialmente, para atender os requisitos da implantação, foi instalado o *Docker* por meio do gerenciador de pacotes do *Ubuntu*. Em seguida, a partir das instruções disponibilizadas na documentação do BDC-STAC, foi implantado um banco de dados *PostgreSQL* preparado com o *Brazil Data Cube Catalog Module*[3]. O banco foi carregado com o exemplo mínimo, em formato JSON, da coleção *sentinel-2* disponibilizada no repositório público do BDC-Catalog[4]. Foi utilizada a versão mais recente disponível tanto para o BDC-STAC quanto para o BDC-Catalog, 1.0.2.

Posteriormente à adequação dos requisitos, de acordo com a documentação do BDC-STAC, foi construída uma Imagem *Docker* nomeada *bdc-stac* usando o comando *docker build*, como demonstrado no código 1.

```
1 docker build --no-cache -t bdc-stac:1.0.2 .
```

Código 1 - Comando, adaptado da documentação do BDC-STAC, usado para construir a imagem *Docker bdc-stac*.

Em seguida, foi criada a rede *bdc_net*, como demonstrado no Código 2, para os contêineres se comunicarem.

[1] https://bdc-stac.readthedocs.io/en/latest/deploy.html
[2] https://github.com/brazil-data-cube/bdc-stac
[3] https://bdc-catalog.readthedocs.io/en/latest/
[4] https://raw.githubusercontent.com/brazil-data-cube/bdc-catalog/master/examples/fixtures/sentinel-2.json

```
docker network create bdc_net

docker network connect bdc_net bdc_pg
```

Código 2 - Comandos para criar a rede *bdc_net* e conectar o contêiner do banco de dados nela.

Por fim, como demonstrado no Código 3, o contêiner com o serviço BDC-STAC foi criado com o comando *docker run*. O contêiner foi adicionado à rede *bdc_net* e conectado ao banco de dados. O serviço foi publicado na porta 8080 do hospedeiro.

```
docker run --detach \
  --name bdc-stac \
  --publish 127.0.0.1:8080:5000 \
  --network=bdc_net \
  --env SQLALCHEMY_DATABASE_URI= \
  "postgresql://postgres:postgres@localhost:5432/bdc_catalog" \
  --env BDC_STAC_BASE_URL="http://localhost:8080" \
  --env BDC_STAC_FILE_ROOT="http://localhost:8081" \
  bdc-stac:1.0.2
```

Código 3 - Comando para criar o contêiner do serviço BDC-STAC a partir da Imagem *Docker* construída anteriormente.

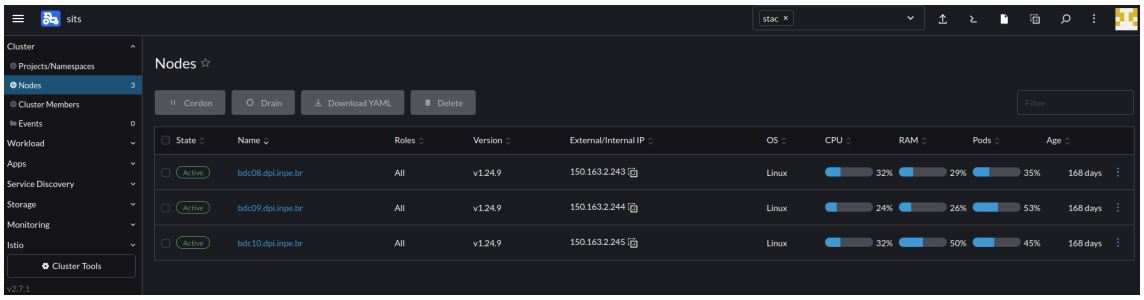
3.2 Configuração do ambiente computacional

O ambiente computacional disponibilizado para realizar os experimentos é composto por quatro máquinas virtuais (VMs) nomeadas *bdc07*, *bdc08*, *bdc09* e *bdc10*. As quatro VMs estão equipadas com 16 GB de memória e 110 GB de armazenamento. Em relação ao número de vCPUs, a VM *bdc10* está equipada com oito e as VMs *bdc07*, *bdc08* e *bdc09* estão equipadas com quatro cada. O sistema operacional (SO) instalado nas VMs é o *Ubuntu* 20.04 com *kernel* na versão 5.4.

O *cluster Kubernetes* instalado nas VMs está na versão 1.24.9 e emprega o contêiner *runtime Docker* na versão 23.0.1. Cada VM é um nó diferente do *cluster* e, portanto, há quatro nós: um *master* que está na VM *bdc07* e três *workers* que estão nas VMs *bdc08*, *bdc09* e *bdc10*. Para controlar o *cluster* através de uma interface gráfica, é possível utilizar o *Rancher* que também está instalado nas VMs. Além

disso, o *Rancher* facilita a configuração e atualização de plataformas de orquestração. A solução para o armazenamento do *cluster* é o *Longhorn*, uma ferramenta de armazenamento persistente e distribuído. Ela facilita a criação e recuperação de *snapshots* de volumes.

Figura 3.1 - Visualização da plataforma *Rancher* em um navegador.



Fonte: Produção do próprio autor.

3.3 Implantação do BDC-STAC no *cluster Kubernetes*

Concluídas as primeiras etapas do estudo, foi iniciada a implantação do BDC-STAC no *cluster Kubernetes* disponibilizado na infraestrutura computacional do BDC. Para o BDC-STAC, um serviço que se conecta a um banco de dados PostgreSQL, são necessários *pods* para o serviço e para o banco de dados. Visando isolar os componentes dessa implantação do resto do *cluster*, todos eles foram criados em um *namespace* novo chamado *stac*.

Para configurar os *pods*, foi necessário definir e escrever os manifestos utilizados na implantação do BDC-STAC. O primeiro manifesto escrito, chamado *bdc-stac*, é referente ao serviço BDC-STAC. Nele, foram definidos os objetos de recurso *service* e *deployment*.

No manifesto *bdc-stac*, o objeto *service* mapeia a porta 5000, padrão do STAC, do contêiner para a porta 8080. Apresentado no Código 4.

```yaml
spec:
  ports:
  - name: http
    port: 8080
    targetPort: 5000
```

Código 4 - Seção *ports* do objeto *service* definido em *bdc-stac.yaml*

Nesse exemplo, o número de réplicas do serviço BDC-STAC foi definido igual a uma, mas pode ser facilmente alterado para outra quantidade, como apresentado no Código 5.

```yaml
spec:
  replicas: 1
```

Código 5 - Seção *replica* do objeto *deployment* definido em *bdc-stac.yaml*

A imagem utilizada para o contêiner *bdc-stac* está em um repositório privado e, para não inserir credenciais no manifesto, foi usado o objeto *secrets*. Esse objeto é criado para armazenar informações confidenciais que serão usadas por algum componente do *cluster*.

O comando *docker login* gerou um arquivo chamado *config.json* com o *token* de acesso para o repositório. Usando o *kubectl*, foi possível criar o *secret* a partir do *token* gerado, conforme apresentado no Código 6.

```
docker login registry.dpi.inpe.br

kubectl create secret generic registry-dpi \
--from-file=.dockerconfigjson=<path/to/.docker/config.json> \
--type=kubernetes.io/dockerconfigjson
```

Código 6 - Comandos para criar o *secret registry-dpi*.

O Código 7 demonstra que para fazer o download de uma imagem de um registro privado, o campo *imagePullSecrets* especifica que o *Kubernetes* deve usar o arquivo

registry-dpi.

```
    spec:
      imagePullSecrets:
        - name: registry-dpi
```

Código 7 - Seção *imagePullSecrets* do objeto *deployment* definido em *bdc-stac.yaml*

Um segundo objeto *secrets*, *bdc-stac-secret*, foi criado para armazenar credenciais necessárias ao contêiner. Apresentado no Código 8.

```
kubectl create secret generic bdc-stac-secret \
--from-env-file .env -n stac
```

Código 8 - Comando para criar o *secret bdc-stac-secret*

```
      containers:
      - name: bdc-stac
        image: registry.dpi.inpe.br/brazil-data-cube/bdc-stac:1.0.2
        envFrom:
        - secretRef:
            name: "bdc-stac-secret"
```

Código 9 - Seção *containers* do objeto *deployment* definido em *bdc-stac.yaml*

O segundo manifesto, *postgres*, é referente ao banco de dados *PostgreSQL*. Nele, foram descritos os objetos *service* e *deployment*. No objeto *service*, a porta de destino foi mantida como 5432 como demonstrado no Código 10.

```
spec:
  ports:
  - name: http
    port: 5432
    targetPort: 5432
```

Código 10 - Seção *ports* do objeto *service* definido em *postgres.yaml*

Nesse exemplo, foi definido o número de réplicas igual a um para o banco de dados, apresentado no Código 11.

```
spec:
  replicas: 1
```

Código 11 - Seção *replicas* do objeto *deployment* definido em *postgres.yaml*

Foi definido o volume persistente do banco de dados *PostgreSQL* no objeto *deployment*. No campo de volumes, foi criado um PVC para fazer a requisição por armazenamento. Para isso, foi criado um manifesto exclusivo para o PVC. Apresentado no Código 12.

```
spec:
  volumes:
    - name: postgres-vol
      persistentVolumeClaim:
        claimName: stac-postgres-pvc
```

Código 12 - Seção *volumes* do objeto *deployment* definido em *postgres.yaml*

A imagem de contêiner *post-gis* possibilita executar o PostgreSQL com as extensões do *PostGIS* instaladas. Foi feito o download da versão 15-3.3 da imagem e a senha padrão não foi alterada, como pode ser observado no Código 13. Não foi feita alteração pois o objetivo desse banco de dados consiste em exclusivamente auxiliar na execução de testes.

```yaml
containers:
- name: postgres
  image: postgis/postgis:15-3.3
  env:
  - name: POSTGRES_PASSWORD
    value: "postgres"
```

Código 13 - Seção *containers* do objeto *deployment* definido em *postgres.yaml*

A porta do contêiner também foi mantida padrão, sendo a 5432 para o *PostgreSQL*, como pode ser observado no Código 14.

```yaml
ports:
- name: http
  containerPort: 5432
```

Código 14 - Seção *ports* do objeto *deployment* definido em *postgres.yaml*

O *Kubernetes* checa algumas informações sobre os *pods* através das ferramentas *probes*. O *kubelet* sabe quando reiniciar um *pod* devido a *livenessProbe* e, usando a *readinessProbe*, o *kubelet* sabe quando começar a aceitar tráfego. Tais configurações são demonstradas no Código 15.

```yaml
readinessProbe:
  exec:
    command: ["psql", "-w", "-U", "postgres", "-c", "SELECT 1"]
  initialDelaySeconds: 15
  timeoutSeconds: 2
livenessProbe:
  exec:
    command: ["psql", "-w", "-U", "postgres", "-c", "SELECT 1"]
  initialDelaySeconds: 30
  timeoutSeconds: 2
```

Código 15 - Seção *probe* do objeto *deployment* definido em *postgres.yaml*

A ferramenta *volumeMounts* define onde os volumes são montados no sistema de

aquivos dos contêineres em *pods*. Tais configurações são apresentado no Código 16.

```
volumeMounts:
- mountPath: "/var/lib/postgresql/data"
  name: postgres-vol
  subPath: postgres
```

Código 16 - Seção *volumeMounts* do objeto *deployment* definido em *postgres.yaml*

O terceiro manifesto descreve um objeto *PersistentVolumeClaim* (PVC). Foi requisitado nesse PVC 2 GB de armazenamento montável para leitura e escrita. A classe de armazenamento requisitada foi *longhorn*, o que pode ser observado no Código 17.

```
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  resources:
    requests:
      storage: 2Gi
```

Código 17 - Seção *spec* do objeto PVC definido em *postgres-pvc.yaml*

Por fim, o último manifesto, Código 18, que descreve um objeto *Ingress* utilizando o *NGINX* como *proxy* reverso.

```
metadata:
  name: bdc-stac-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/configuration-snippet: |
      rewrite /stac(.*) /$1 break;
      proxy_set_header X-Stac-Url "https://bdc08.dpi.inpe.br/stac";
```

Código 18 - Seção *annotation* do objeto *Ingress* definido em *ingress.yaml*

Nesse exemplo, as requisições chegam no *host bdc08.dpi.inpe.br*, no caminho */stac* e

são enviadas para o *service bdc-stac-svc*, de acordo com o definido no Código 19.

```
spec:
  ingressClassName: nginx
  rules:
    - host: bdc08.dpi.inpe.br
      http:
        paths:
          - path: /stac
            pathType: Prefix
            backend:
              service:
                name: bdc-stac-svc
                port:
                  name: http
```

Código 19 - Seção *spec* do objeto *Ingress* definido em *ingress.yaml*

Depois de escrever todos os arquivos manifestos necessários para a implantação do *BDC-STAC* no *cluster*, eles foram aplicados com a ferramenta *kubectl* para que, então, os *pods* necessários fossem criados. A sequência de aplicação dos manifestos foi apresentada no Código 20.

```
kubectl apply -f postgres-pvc.yaml
kubectl apply -f postgres.yaml
kubectl apply -f bdc-stac.yaml
kubectl apply -f ingress.yaml
```

Código 20 - Sequência de aplicação dos arquivos manifestos.

4 RESULTADOS

A implantação do BDC-STAC utilizando Docker, proposta na primeira etapa do estudo foi cumprida. Foi possível implantar os dois contêineres necessários (serviço BDC-STAC e banco de dados) sem adversidades. Para testar o funcionamento do serviço BDC-STAC foi utilizado, de acordo com o exemplo demonstrado em sua documentação, o comando curl na porta 8080 do sistema hospedeiro. Esse comando, apresentado no Código 21, retornou, em JSON, os metadados da coleção sentinel-2 que foi carregada no banco de dados.

```
curl localhost:8080
```

Código 21 - Comando para requisitar os metadados do serviço BDC-STAC.

O resultado dessa etapa está parcialmente apresentado no Código 22, demonstrando assim que foi possível utilizar o serviço.

```
{
    "type": "Catalog",
    "description": "Brazil Data Cube Catalog",
    "id": "bdc",
    "stac_version": "1.0.0",
    "links": [
        {
            "href": "http://localhost:8080/",
            "rel": "self",
            "type": "application/json",
            "title": "Link to this document"
        },
```

Código 22 - Resposta da requisição curl na porta 8080 do hospedeiro.

Para a implantação do BDC-STAC no cluster Kubernetes no ambiente computacional do BDC, os arquivos manifestos foram aplicados e os objetos descritos: pods, serviços, volume persistente e Ingress foram implantados no cluster.

A fim de verificar o funcionamento do cluster, foi utilizada a ferramenta kubectl para listar os pods implantados no namespace stac. A príncipio, fixando apenas

uma réplica, eram esperados dois pods, um para o serviço BDC-STAC e outro para o banco de dados PostgreSQL, o que pode ser obtido, como pode ser visto no Código 23.

```
ubuntu@desktop:~$ kubectl get pods -n stac -o wide
NAME                          READY   STATUS    NODE
bdc-stac-5594449d95-zm4sq     1/1     Running   bdc08.dpi.inpe.br
postgres-85874ffb79-f5sxm     1/1     Running   bdc10.dpi.inpe.br
```

Código 23 - Saída do comando para listagem de pods no namespace stac.

Confirmado o status dos pods quando fixado uma réplica, o próximo passo foi aumentar o número de réplicas do serviço BDC-STAC. Para isso, foi alterado o valor no objeto deployment do manifesto bdc-stac. Foi alterado para duas réplicas, conforme Código 24.

```
spec:
  replicas: 2
```

Código 24 - Seção *replica* do objeto *deployment* definido em *bdc-stac.yaml*

```
ubuntu@desktop:~$ kubectl get pods -n stac -o wide
NAME                          READY   STATUS    NODE
bdc-stac-5594449d95-cjmhd     1/1     Running   bdc09.dpi.inpe.br
bdc-stac-5594449d95-zm4sq     1/1     Running   bdc08.dpi.inpe.br
postgres-85874ffb79-f5sxm     1/1     Running   bdc10.dpi.inpe.br
```

Código 25 - Saída do comando para listagem de pods no namespace stac.

Por fim, o número de réplicas do serviço BDC-STAC foi alterado novamente, dessa vez, para quatro, conforme Código 26, resultando na listagem que pode ser observada no Código 27.

```
spec:
  replicas: 4
```

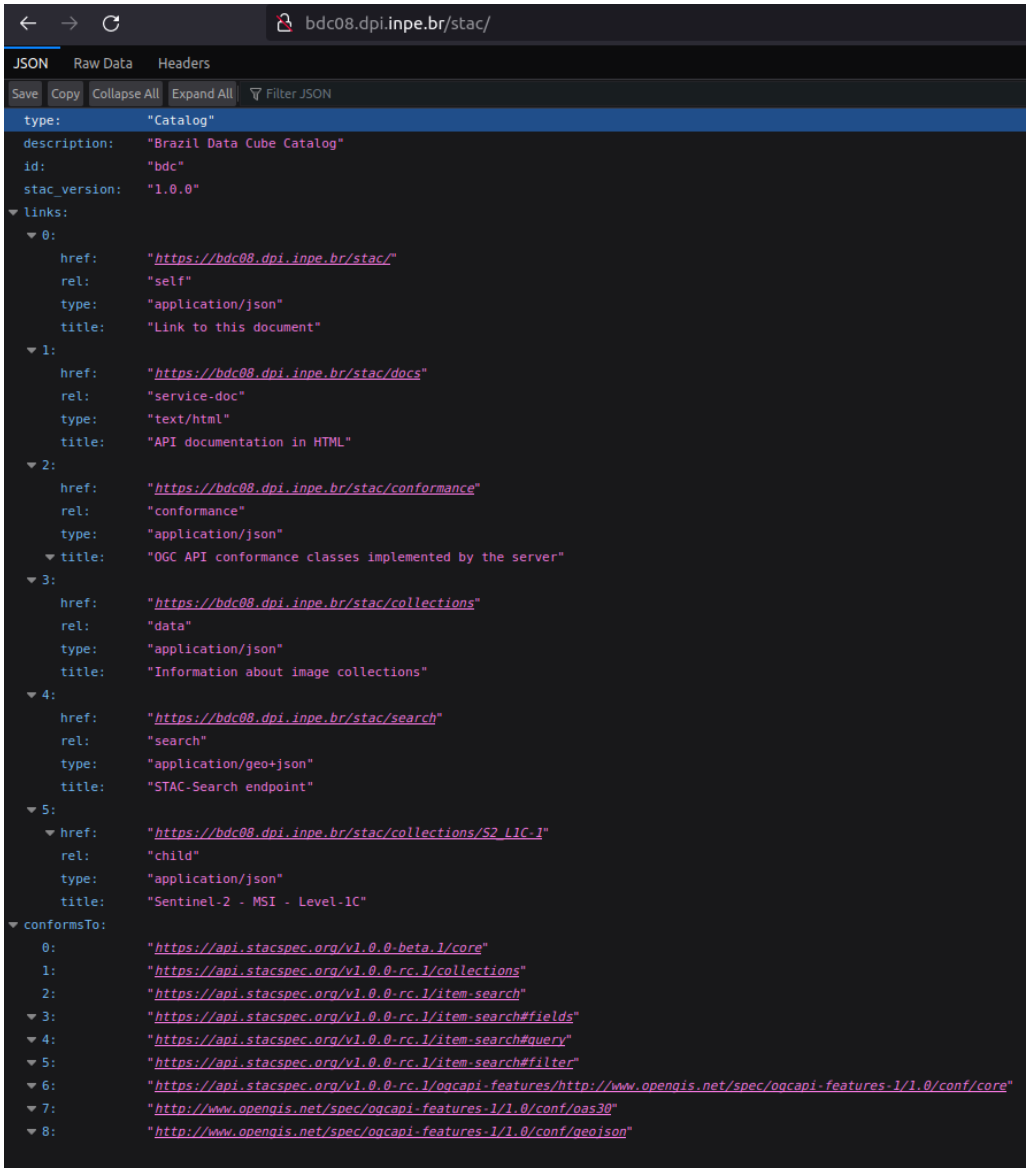
Código 26 - Seção *replica* do objeto *deployment* definido em *bdc-stac.yaml*

```
ubuntu@desktop:~$ kubectl get pods -n stac -o wide
NAME                          READY   STATUS    NODE
bdc-stac-5594449d95-7ddgh     1/1     Running   bdc10.dpi.inpe.br
bdc-stac-5594449d95-hlq75     1/1     Running   bdc09.dpi.inpe.br
bdc-stac-5594449d95-lc98x     1/1     Running   bdc08.dpi.inpe.br
bdc-stac-5594449d95-zm4sq     1/1     Running   bdc08.dpi.inpe.br
postgres-85874ffb79-f5sxm     1/1     Running   bdc10.dpi.inpe.br
```

Código 27 - Saída do comando para listagem de pods no namespace stac.

Portanto, os testes funcionaram integralmente. Todos os pods foram executados e, em nós distintos. Também foi possível verificar o funcionamento do BDC-STAC pelo navegador na URL fornecida pelo Ingress. O retorno é uma página com os metadados (JSON) do catálogo que foi carregado no PostgreSQL. Dessa forma, o Ingress e o volume persistente também estão funcionando.

Figura 4.1 - Visualização do JSON retornado pelo BDC-STAC em um navegador.



Fonte: Produção do próprio autor.

5 CONCLUSÃO

Considerando a revisão bibliográfica realizada, notou-se que a área de geotecnologia está lidando com uma demanda crescente por armazenamento e processamento de dados geoespaciais. Dessa forma, é essencial a adoção de novas tecnologias voltadas para maior eficiência no uso de recursos computacionais. As tecnologias de virtualização, dentre elas, as máquinas virtuais e os contêineres estão auxiliando no desenvolvimento de ambientes modulares e com divisão eficiente de recursos físicos que podem ser compartilhados entre múltiplas aplicações. Inclusive sendo possível o uso de orquestradores de contêineres para melhor gerir aplicações e facilitar a escalabilidade.

Tendo em vista a importância das tecnologias de virtualização, o estudo em áreas que integram elas é cada vez mais relevante. Esse projeto teve como objetivo prático a implantação de um serviço geoespacial, BDC-STAC, utilizando duas arquitetura complementares, os contêineres e os orquestradores. O BDC-STAC se mostrou apto a ser instanciados em contêineres *Dockers* seguindo a sua documentação. Então, a implantação do serviço foi estendida para um conjunto de contêineres gerenciados pelo orquestrador Kubernetes, o que também se mostrou possível, inclusive com fácil escalabilidade para variação do número de réplicas.

Assim, outros serviços da plataforma BDC, em especial os de visualização de dados, também podem se beneficiar de melhorias fornecidas por uma implantação em ambiente *Kubernetes*.

REFERÊNCIAS BIBLIOGRÁFICAS

BARIK, R. K.; LENKA, R. K.; RAO, K. R.; GHOSE, D. Performance analysis of virtual machines and containers in cloud computing. In: IEEE. **2016 international conference on computing, communication and automation (iccca)**. Bhubaneswar, India, 2016. p. 1204–1210. 8, 9

BAUMANN, P. The OGC web coverage processing service (WCPS) standard. **Geoinformatica**, Springer, v. 14, p. 447–479, 2010. 5

BAUMANN, P.; ROSSI, A. P.; BELL, B.; CLEMENTS, O.; EVANS, B.; HOENIG, H.; HOGAN, P.; KAKALETRIS, G.; KOLTSIDA, P.; MANTOVANI, S. et al. Fostering cross-disciplinary earth science through datacube analytics. **Earth Observation Open Science and Innovation**, Springer International Publishing, p. 91–119, 2018. 6

BODENHAMER, D. J. Beyond gis: Geospatial technologies and the future of history. In: **History and GIS: Epistemologies, considerations and reflections**. New York: Springer, 2012. p. 1–13. 5

BOETTIGER, C. An introduction to docker for reproducible research. **ACM SIGOPS Operating Systems Review**, ACM New York, NY, USA, v. 49, n. 1, p. 71–79, 2015. 2

CHAE, M.; LEE, H.; LEE, K. A performance comparison of linux containers and virtual machines using docker and kvm. **Cluster Computing**, Springer, v. 22, n. Suppl 1, p. 1765–1775, 2019. 8

FERREIRA, K. R.; QUEIROZ, G. R.; VINHAS, L.; MARUJO, R. F. B.; SIMOES, R. E. O.; PICOLI, M. C. A.; CAMARA, G.; CARTAXO, R.; GOMES, V. C. F.; SANTOS, L. A.; SANCHEZ, A. H.; ARCANJO, J. S.; FRONZA, J. G.; NORONHA, C. A.; COSTA, R. W.; ZAGLIA, M. C.; ZIOTI, F.; KORTING, T. S.; SOARES, A. R.; CHAVES, M. E. D.; FONSECA, L. M. G. Earth Observation Data Cubes for Brazil: Requirements, Methodology and Products. **Remote Sensing**, v. 12, n. 24, p. 4033, dec 2020. ISSN 2072-4292. Disponível em: <<https://www.mdpi.com/2072-4292/12/24/4033>>. 1, 6

FURTADO, A. d. S. Disponibilização de geoinformação utilizando plataformas livres: Webgis do parque nacional da restinga de jurubatiba. In: Engenharia e ciências ambientais: contribuições à gestão ecossistêmica . . . , 2018. 5

GIULIANI, G.; CHATENOUX, B.; BONO, A. D.; RODILA, D.; RICHARD, J.-P.; ALLENBACH, K.; DAO, H.; PEDUZZI, P. Building an earth observations data cube: lessons learned from the swiss data cube (sdc) on generating analysis ready data (ard). **Big Earth Data**, Taylor & Francis, v. 1, n. 1-2, p. 100–117, 2017. 1, 6

JAWARNEH, I. M. A.; BELLAVISTA, P.; BOSI, F.; FOSCHINI, L.; MARTUSCELLI, G.; MONTANARI, R.; PALOPOLI, A. Container orchestration engines: A thorough functional and performance comparison. In: IEEE. **ICC**

2019-2019 IEEE International Conference on Communications (ICC). [S.l.], 2019. p. 1–6. 2

KUBERNETES. 2023. Acessado: 13 junho 2023. Disponível em: <<https://kubernetes.io/>>. 10, 11, 12

MCCARTY, S. **A Practical Introduction to Container Terminology**. 2018. Acessado: 10 junho 2023. Disponível em: <<https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction>>. 9, 10

RISTA, A.; AJDARI, J.; ZENUNI, X. Cloud computing virtualization: A comprehensive survey. In: IEEE. **2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)**. Opatija, Croatia, 2020. p. 462–472. 1

RODRIGUEZ, M. A.; BUYYA, R. Container-based cluster orchestration systems: A taxonomy and future directions. **Software: Practice and Experience**, Wiley Online Library, v. 49, n. 5, p. 698–719, 2019. 2, 10, 11

ROSSI, F.; CARDELLINI, V.; PRESTI, F. L.; NARDELLI, M. Geo-distributed efficient deployment of containers with kubernetes. **Computer Communications**, Elsevier, v. 159, p. 161–174, 2020. 10, 11

SCHOLTEN, H. J.; VELDE, R.; MANEN, N. van. **Geospatial Technology and the Role of Location in Science**. New York: Springer Science & Business Media, 2009. 1, 5

SHARMA, P.; CHAUFOURNIER, L.; SHENOY, P.; TAY, Y. Containers and virtual machines at scale: A comparative study. In: ACM. **Proceedings of the 17th international middleware conference**. Singapore, 2016. p. 1–13. 1, 8

SHAW, S.-L.; YU, H.; BOMBOM, L. S. A space-time gis approach to exploring large individual-based spatiotemporal datasets. **Transactions in GIS**, Wiley Online Library, v. 12, n. 4, p. 425–441, 2008. 5

SIDDIQUI, T.; SIDDIQUI, S. A.; KHAN, N. A. Comprehensive analysis of container technology. In: IEEE. **2019 4th International Conference on Information Systems and Computer Networks (ISCON)**. Mathura, INDIA, 2019. p. 218–223. 1

STAC. **SpatioTemporal Asset Catalogs**. 2023. [Online; accessed 7-August-2023]. Disponível em: <<https://stacspec.org/en>>. 1, 7

SUDMANNS, M.; TIEDE, D.; LANG, S.; BERGSTEDT, H.; TROST, G.; AUGUSTIN, H.; BARALDI, A.; BLASCHKE, T. Big earth data: disruptive changes in earth observation data management and analysis? **International Journal of Digital Earth**, Taylor & Francis, v. 13, n. 7, p. 832–850, 2020. 5, 6

ZAGLIA, M. C. **Catalogação de imagens de satélites de sensoriamento remoto voltada para acesso a cubos de dados de observação da Terra**.

59 p. Dissertação (Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2020. 7

ZAGLIA, M. C.; VINHAS, L.; QUEIROZ, G. R. de; SIMÕES, R. E. O. Catalogação de metadados do cubo de dados do brasil com o spatiotemporal asset catalog. In: **GEOINFO**. São José dos Campos, Brazil: [s.n.], 2019. p. 280–285. 1, 8