sid.inpe.br/mtc-m21d/2023/03.30.19.42-TDI

BRAZIL DATA CUBE WORKFLOW ENGINE: A TOOL FOR BIG EARTH OBSERVATION DATA PROCESSING

Vitor Conrado Faria Gomes

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Karine Reis Ferreira Gomes, and Gilberto Ribeiro de Queiroz, approved in March 29, 2023.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/48QKERL>

INPE
São José dos Campos
2023

PUBLISHED BY:

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

sid.inpe.br/mtc-m21d/2023/03.30.19.42-TDI

BRAZIL DATA CUBE WORKFLOW ENGINE: A TOOL FOR BIG EARTH OBSERVATION DATA PROCESSING

Vitor Conrado Faria Gomes

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Karine Reis Ferreira Gomes, and Gilberto Ribeiro de Queiroz, approved in March 29, 2023.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/48QKERL>

INPE
São José dos Campos
2023

Cataloging in Publication Data

Gomes, Vitor Conrado Faria.

G585b Brazil Data Cube Workflow Engine: a tool for big earth observation data processing / Vitor Conrado Faria Gomes. – São José dos Campos : INPE, 2023.

xx + 97 p. ; (sid.inpe.br/mtc-m21d/2023/03.30.19.42-TDI)

Thesis (Doctorate in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2023.

Guiding : Drs. Karine Reis Ferreira Gomes, and Gilberto Ribeiro de Queiroz.

1. Big data. 2. Directed acyclic graphs. 3. Open data cube. 4. OpenEO. 5. Dagster. I.Title.

CDU 004.6:528.8

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DEFESA FINAL DE TESE VITOR CONRADO FARIA GOMES BANCA Nº 042/2023, REGISTRO 113654/2017

No dia 29 de março de 2023, as 09h, por teleconferência, o(a) aluno(a) mencionado(a) acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O(A) aluno(a) foi APROVADO(A) pela Banca Examinadora, por unanimidade, em cumprimento ao requisito exigido para obtenção do Título de Doutor em Computação Aplicada, com a exigência de que o trabalho final a ser publicado deverá incorporar as correções sugeridas pela Banca Examinadora, com revisão pelo(s) orientador(es).

Título: "Brazil Data Cube Workflow Engine: a tool for big Earth Observation data processing".

Membros da Banca:

Dr. Thales Sehn Korting – Presidente – INPE
Dra. Karine Reis Ferreira Gomes – Orientadora – INPE
Dr. Gilberto Ribeiro de Queiroz – Orientador – INPE
Dr. Cláudio Clemente Faria Barbosa - Membro Interno - INPE
Dr. Cláudio Elízio Calazans Campelo - Membro Externo - UFCG
Dr. Vinícius Vielmo Cogo - Membro Externo - ULisboa

Documento assinado eletronicamente por **CLAUDIO ELIZIO CALAZANS CAMPELO (E)**, **Usuário Externo**, em 05/04/2023, às 23:59 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Cláudio Clemente Faria Barbosa**, **Tecnologista**, em 13/04/2023, às 09:06 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.mcti.gov.br/verifica.html, informando o código verificador **10933930** e o código CRC **7B79B348**.

Referência: Processo nº 01340.002139/2023-02 SEI nº 10933930

“The best race car drivers understand how their cars work. The best architects know how carpenters, bricklayers, and electricians do their jobs. And the best programmers know how the hardware they are programming does computation”.

MARK L. CHANG

em *“Reconfigurable Computing”*, 2008

ACKNOWLEDGEMENTS

First, I would like to thank my family, especially my parents, Miriam Helena, and Mario Sergio and my wife, Luciane.

Special thanks to my advisors, Dr. Karine Ferreira and Dr. Gilberto Queiroz, who guided me, motivated me, and knew how to say the right words in difficult moments.

I would also like to thank the people who dedicate part of their time to creating software or free intellectual works, which directly or indirectly supported the development of this work.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

ABSTRACT

Earth Observation (EO) satellites have produced large amounts of geospatial data that are freely available to society and researchers. Handling these data often exceeds the capabilities of the hardware and software traditionally used for storing and processing EO data. This scenario presents challenges for traditional Spatial Data Infrastructure (SDI) to properly store, process, disseminate, and analyze big data sets. To meet these demands, new technologies based on cloud computing and distributed systems, such as matrix database systems, MapReduce systems, and web services, have been proposed and developed. These technologies are now being integrated into leading-edge platforms to support a new generation of SDI for big EO data. These platforms have different characteristics in terms of governance, technologies used, data access, infrastructure abstractions, data processing, and flexibility to extend their functionality. In general, we observed that the greater the degree of abstraction given to the scientist, the greater the difficulty in providing flexibility in data-processing approaches. This thesis contributes to the area of spatial data infrastructure through the evaluation and analysis of available EO data processing and analysis platforms as well as a server-side EO data processing architecture that provides an abstraction of access and processing of EO data for users and the possibility of including algorithms and access and processing techniques by SDI maintainers. The main idea was to build a framework based on workflow orchestration tools integrated with a high-level API for user interaction. This tool allows the configuration of processes and the extension of previously defined data models. Furthermore, the interface between the processing services and the user is executed through the OpenEO API, which establishes a standard for accessing, manipulating and processing EO data. The architecture proposed in this thesis was implemented and applied in two case studies.

Keywords: Big Data. Directed Acyclic graphs. Open Data Cube. OpenEO. Dagster.

BRAZIL DATA CUBE WORKFLOW ENGINE: UMA FERRAMENTA PARA PROCESSAMENTO DE GRANDES VOLUMES DE DADOS DE OBSERVAÇÃO DA TERRA

RESUMO

Satélites de observação da Terra (*Earth Observation* - EO) têm produzido grandes quantidades de dados geoespaciais que estão disponíveis gratuitamente para a sociedade e pesquisadores. Frequentemente, a manipulação desses dados excedem as capacidades de hardware e software tradicionalmente usados para o armazenamento e processamento de dados de EO. Este cenário traz desafios para as infraestruturas tradicionais de dados espaciais (SDI) para armazenar, processar, disseminar e analisar adequadamente esses conjuntos de big data. Para atender a essas demandas, novas tecnologias foram propostas e desenvolvidas, baseadas em computação em nuvem e sistemas distribuídos, como sistemas de banco de dados matriciais, sistemas MapReduce e serviços web, para acessar e processar esses volumes de dados. Atualmente, essas tecnologias vêm sendo integradas em plataformas de ponta para suportar uma nova geração de SDI para grandes volumes de dados de EO. Essas plataformas apresentam diferentes características em relação à governança, tecnologias utilizadas, acesso aos dados, abstrações de infraestrutura, dados e processamento e quanto à flexibilidade de extensão de suas funcionalidades. De maneira geral, observamos que quanto maior o grau de abstração entregue ao cientista, maior a dificuldade em fornecer flexibilidade nas abordagens de processamento de dados. Essa tese contribui para a área de infraestrutura de dados espaciais por meio da avaliação e análise de plataformas de processamento e análise de dados de EO disponíveis e pela proposição de uma arquitetura de processamento de dados de EO no lado do servidor que fornece, aos usuários, abstração de acesso e processamento de dados. Essa arquitetura é estruturada na forma de um framework baseado em ferramentas de orquestração de workflows, integrado com uma API de alto nível para a interação com os usuários. Essa ferramenta permite a configuração de processamentos e a extensão dos modelos de dados previamente definidos. Além disso, a interface entre os serviços de processamento e o usuário é feita por meio da OpenEO API, a qual estabelece um padrão para o acesso, manipulação e processamento de dados de EO. A arquitetura proposta nesta tese foi implementa e aplicada em dois estudos de caso.

Palavras-chave: Grafos acíclicos dirigidos. Open Data Cube. OpenEO. Grandes volumes de dados. Dagster.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AGDC	–	Australian Geoscience Data Cube
API	–	Application Programming Interface
ARD	–	Analysis Ready Data
AWS	–	Amazon Web Services
BDC	–	Brazil Data Cube
BDC-WE	–	Brazil Data Cube Workflow Engine
CEOS	–	Committee on Earth Observation Satellite
CRS	–	Coordinate Reference System
CSIRO	–	Commonwealth Scientific and Industrial Research Organization
CSW	–	Catalog Service for Web
DBMS	–	Database Management Systems
EO	–	Earth Observation
EODC	–	Earth Observation Data Cube
GA	–	Geoscience Australia
GDAL	–	Geospatial Data Abstraction Library
GEE	–	Google Earth Engine
GFS	–	Google File System
GIS	–	Geographic Information System
HDFS	–	Hadoop Distributed File System
IDE	–	Integrated Development Environment
JEODPP	–	JRC EO Data and Processing Platform
JRC	–	Joint Research Centre
ODC	–	Open Data Cube
OGC	–	Open Geospatial Consortium
RDBMS	–	Relational DBMS
REST	–	Representational State Transfer
SDI	–	Spatial Data Infraestructure
SEPAL	–	System for EO, data access, processing & analysis for land monitoring
SH	–	Sentinel Hub
SITS	–	Satellite Image Time Series Analysis for Earth Observation Data Cubes
STAC	–	SpatioTemporal Asset Catalogs
TMS	–	Tile Map Service
UDF	–	User Defined Function
USGS	–	United States Geological Survey
WCPS	–	Web Coverage Processing Service
WCS	–	Web Coverge Service
WFS	–	Web Feature Service
WLTS	–	Web Land Trajectory Service
WMS	–	Web Map Service
WMTS	–	Web Map Tile Service
WTSS	–	Web Time Series Service

CONTENTS

Page

1 INTRODUCTION

The scientific community has increasingly made use of the wide availability of large Earth Observation (EO) datasets to advance the understanding of processes on Earth, monitoring environmental changes, detecting risks, and analyzing urban occupation (CAMARA et al., 2014; APPEL et al., 2018). However, dealing with these massive datasets still represents a great challenge for extracting their full potential and value (APPEL; PEBESMA, 2019), implying that in practice, only a small part of the available data is effectively used for scientific research and operational applications (CAMARA et al., 2014).

To address these challenges, the EO community has been developing new technologies, such as platforms for big EO data. These platforms are computational solutions offering a range of functionalities for managing, storing, and accessing big EO data. These technologies allow server-side processing, eliminating the need to download massive amounts of EO datasets. In addition, they provide a certain level of data and processing abstractions that are useful to EO community users and researchers (GOMES et al., 2020). These platforms integrate different types of technologies, Application Programming Interfaces (APIs), and web services, resulting in a more comprehensive solution for managing and analyzing big EO data. The *Moving Code* paradigm is one of the concepts used in these systems. This paradigm appears as an alternative to the classic client-server model, where data needs to be transferred between servers and clients to be processed. In the *Moving Code* approach, the executable content (or code) is moved closer to the other resources involved in the computation, aiming for greater efficiency. A frequent application of this approach is sending the code to be executed on the servers where the data is located. In this situation, only the analysis results are moved between client and server (MÜLLER, 2016). Some platforms for big EO data that provide some level of support to users for server-side data processing are the Google Earth Engine (GEE) (GOOGLE, 2020), OpenEO (OPENEO, 2022), Sentinel Hub (SINERGISE, 2020a), and Open Data Cube (ODC) (OPEN DATA CUBE, 2022b).

In the Brazilian context, the Brazil Data Cube (BDC) project is an initiative, created in 2019 by the National Institute for Space Research (INPE), which leads the development of a platform for big EO data called Brazil Data Cube. The BDC project has four main objectives (FERREIRA et al., 2020): (1) create Analysis Ready Data (ARD) image collections from spatial medium-resolution remote sensing images (10 to 64 m) for the whole Brazilian territory; (2) model these ARD images as multidimensional data cubes with three or more dimensions that include space, time, and spectral-derived properties, mainly to support image time series analysis; (3) use, propose, and develop big data technologies, such as cloud computing and distributed processing environments, to create, store, and process these data cubes; and (4) create land use and cover information, for Brazil, from these data cubes using satellite image time series analysis, machine learning methods, and image processing procedures.

Despite the great advances already made in this area, the technologies for processing and analyzing large sets of OE data are still in constant evolution, and there is no predominant solution, although there are solutions that stand out (GOMES et al., 2020). For this reason, exploring, proposing, and developing solutions to address the challenges of processing large Earth Observation (EO) datasets are still significant for the geoprocessing community.

1.1 Our proposal

As part of the BDC project initiative, this thesis contributes to the proposal, design and development of a system for big EO data processing on the server side. This system is part of BDC platform and is based on the OpenEO API, which has recently been established as a standard adopted in several platforms for processing EO data (EUROPEAN SPACE AGENCY, 2022).

Our scientific question is: how to design and implement a tool for big EO data processing that allows users and developers of the BDC project to describe sequences of processes to be efficiently executed in the project server-side infrastructure?

Our hypothesis is that for processing big EO data, it is necessary to provide users with an environment for the execution of processing and analysis on the server side, avoiding the need to move data. In addition, we consider that the integration of existing open technologies can facilitate the development of this solution. The flexibility to include new algorithms or customize existing processing, which is important for BDC project developers, can be provided by building this environment in the form of a framework that can be configured and adapted to the project's needs.

To answer our scientific question using our hypotheses, this thesis presents a complete study and review of existing technologies for big EO data processing and a proposal for a system called BDC-Workflow Engine (BDC-WE) for the BDC platform. This system uses technologies for orchestrating processes described by directed acyclic graphs and integrates the OpenEO API to allow the submission and control of processes by the users. A prototype of the BDC-WE was implemented as part of the BDC platform and two case studies were developed to demonstrate its potential.

1.2 Contributions

The main contributions of this thesis are:

I) Study and review of distinct technologies for managing, processing, and analysis of big EO data;

II) A complete review and comparative analysis of the main platforms for big EO data management and analysis;

III) The integration between the BDC platform and the Open Data Cube framework, expanding the available tools to disseminate, process and analyze the BDC data products;

IV) The proposal of a system architecture to allow the execution of user processing in the infrastructure of the BDC project; and

V) The implementation of a prototype of this system called BDC-Workflow Engine, as part of the BDC platform, for processing workflows on the server-side with data access and interaction through an OpenEO API.

The materialization of these contributions can be found in articles and repositories of code and documentation produced by the author of this thesis, which are listed below. Indexes are included at the end of each element of these lists, for example **(I, II)**, to relate the contribution to the work produced.

These contributions can be found in five articles in which the author of this thesis appears as the first author.

a) **GOMES, V. C. F.**; QUEIROZ, G. R.; FERREIRA, K. R.; BARBOSA, C. C. F.; PEBESMA, E.. Brazil Data Cube Workflow Engine: a tool for big Earth Observation data processing. Unsubmitted manuscript on the date of writing of this thesis. **(IV, V)**.

b) **GOMES, V. C. F.**; CARLOS, F. M.; QUEIROZ, G. R.; FERREIRA, K. R.; SANTOS, R.. Accessing and processing brazilian Earth Observation

Data Cubes with the Open Data Cube platform. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, v. V-4-2021, p. 153-159, 2021. **(III)**.

c) **GOMES, V. C. F.**; QUEIROZ, G. R.; FERREIRA, K. R.. An Overview of Platforms for big Earth Observation Data Management and Analysis. Remote Sensing, v. 12, p. 2, 2020. **(I, II)**.

d) **GOMES, V. C. F.**; QUEIROZ, G. R.; FERREIRA, K. R.; SATO, L. Y.; SANTOS, R. D. C.. Um ambiente para análise exploratória de grandes volumes de dados geoespaciais: explorando risco de fogo e focos de queimadas. In: XVIII Brazilian Symposium on Geoinformatics (GeoInfo 2017), 2017, Salvador. XVIII Brazilian Symposium on Geoinformatics Proceedings, v. 1. p. 1, 2017. **(I)**.

e) **GOMES, V. C. F.**; SATO, L. Y.; QUEIROZ, G. R.; VINHAS, L.; FERREIRA, K. R.. Gerenciamento de nuvem de pontos em SGBD: avaliando a extensão PointCloud para PostgreSQL. In: XVII Brazilian Symposium on Geoinformatics (GeoInfo 2016), 2016, Campos do Jordão. Brazilian Symposium on Geoinformatics Proceedings, v. 1, 2016. **(I)**.

These contributions also materialized in the form of documents, scripts, and libraries available in the following repositories:

a) **bdc-we**: the BDC-WE prototype. Available in <https://github.com/brazil-data-cube/bdc-we>. **(IV, V)**.

b) **bdc-we-template**: a BDC-WE boilerplate project. Available in <https://github.com/brazil-data-cube/bdc-we-template>. **(IV, V)**.

c) **dagster_graphql_client**: a Python client for working with Dagster GraphQL servers. Available in <https://github.com/vconrado/dagster_graphql_client>. **(IV, V)**.

d) **bdc-odc**: a collection of scripts and tools to facilitate the deployment and maintenance of an ODC instance. Available in <https://github.com/brazil-data-cube/bdc-odc>. **(III)**.

e) **stac2odc**: a tool to facilitate indexing data in an Open Data Cube (ODC) instance using the information provided by STAC catalogs. Available in <https://github.com/brazil-data-cube/stac2odc>. **(III, V)**.

1.2.1 Related contributions

During the development of the research addressed in this document, other contributions were made to topics related to the main subject of this thesis. These contributions can be found in the following articles, in which the author of this thesis appears as a co-author:

a) FLORES JUNIOR, R.; **GOMES, V. C. F.**; FERREIRA, K. R.; QUEIROZ, G. R.; BONNET, M.. A QGIS plugin for BONDS project: integrating field data with geographical, remote sensing and health information. In: XXIII Brazilian Symposium on Geoinformatics (GEOINFO 2022), 2022, São José dos Campos. Proceedings of the XXIII Brazilian Symposium on Geoinformatics (GEOINFO 2022), v. 1. p. 1-1, 2022. **(I)**.

b) CARLOS, F. M.; **GOMES, V. C. F.**; QUEIROZ, G. R.; SOUZA, F. C.; FERREIRA, K. R.; SANTOS, R.. Integrating Open Data Cube and Brazil Data Cube Platforms for Land Use and Cover Classifications. RBC. REVISTA BRASILEIRA DE CARTOGRAFIA (ONLINE), v. 73, p. 1036-1047, 2021. **(III)**.

c) CARLOS, F. M.; **GOMES, V. C. F.**; QUEIROZ, G. R.; FERREIRA, K. R.; SANTOS, R. D. C.. Integração dos ambientes Brazil Data Cube e Open Data Cube. In: GEOINFO 2020 - Brazilian Symposium on Geoinformatics, 2020, São José dos Campos. GeoInfo 2020 Proceedings, v. 1. p. 1, 2020. **(III)**.

d) 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, p. 4033, 2020. **(I, III)**.

e) FERREIRA, K. R.; QUEIROZ, G. R.; CAMARA, G.; SOUZA, R. C. M.; VINHAS, L.; MARUJO, R. F. B.; SIMOES, R. E. O.; NORONHA, C. A. F.; COSTA, R. W.; ARCANJO, J. S.; **GOMES, V. C. F.**; ZAGLIA, M. C.. Using Remote Sensing Images and Cloud Services on Aws to Improve Land Use and Cover Monitoring. In: 2020 IEEE Latin American GRSS &

ISPRS Remote Sensing Conference (LAGIRS), 2020, Santiago. 2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS). p. 558, 2020. **(I)**.

f) ZIOTI, F.; **GOMES, V. C. F.**; LLAPA, E.; QUEIROZ, G. R.; FERREIRA, K. R.. Um ambiente para acesso e análise de trajetórias de uso e cobertura da Terra. In: XIX Simpósio Brasileiro de Sensoriamento Remoto, 2019, Santos. Anais do XIX Simpósio Brasileiro de Sensoriamento Remoto, 2019. **(I)**.

g) SANCHEZ, A.; VINHAS, L.; QUEIROZ, G. R.; SIMOES, R.; **GOMES, V. C. F.**; ASSIS, L. F. F. G.; LLAPA, E.; CAMARA, G.. Reproducible geospatial data science: Exploratory Data Analysis using collaborative analysis environments. In: Brazilian Symposium on GeoInformatics, 2017, Salvador/BA. Proceedings, p. 7-16, 2017. **(I)**.

In addition to these articles, the following set of software, scripts, and documents related to the objectives of this work were produced by the author of this thesis:

a) **scidb_clusterfier**: a collection of scripts to facilitate a deployment of a SciDB cluster. Available in <https://github.com/vconrado/scidb_clusterfier>. **(I)**.

b) **chunkfier**, **interleaver** and **modis2scidb**: a collection of scripts to assist in preparing and loading EO data into a SciDB instance. Available in <https://github.com/vconrado/chunkfier>, <https://github.com/vconrado/interleaver>, and <https://github.com/vconrado/modis2scidb>. **(I)**.

c) **avaliacao-sgbd-m**: a collection of scripts and documentation used to automate benchmarking of Array DBMS SciDB, RasDaMan, and TileDB. Available in <https://github.com/vconrado/avaliacao-sgbd-m>. **(I)**.

d) **hadoop-docker**: a collection of scripts to assist the management of a Hadoop cluster with multiple nodes. Available in <https://github.com/vconrado/hadoop-docker>. **(I)**.

e) **simple_geo.py**: a Python client for integrating data from WFS and WTSS. Available in <https://github.com/vconrado/simple_geo.py>. **(I)**.

Another contribution related to the theme of this thesis is the dissemination of knowledge acquired during the development of this research. This was done in the form of classes taught or through elaboration or collaboration in short courses:

a) **GOMES, V. F. G.**; QUEIROZ, G. R.. Novas Abordagens em Banco de Dados não Relacionais para Dados Geográficos e Plataformas de big Data para Dados de Observação da Terra. Fundamentals of Geoprocessing Course (SER-300) of the graduate program in Remote Sensing at INPE. Class, 2019, 2020, 2021, and 2022. Available in <http://wiki.dpi.inpe.br/doku.php?id=ser300:aulas_2022>. **(I, II)**.

b) QUEIROZ, G. R.; **GOMES, V. F. G.**. Arquiteturas de SIG & Banco de Dados Geográficos. Fundamentals of Geoprocessing Course (SER-300) of the graduate program in Remote Sensing at INPE. Class, 2019, 2020, 2021, and 2022. Available in <http://wiki.dpi.inpe.br/doku.php?id=ser300:aulas_2022>. **(I, II)**.

c) SANTOS, R.; MEDEIROS, F.; BITTENCOURT, O. QUEIROZ, G. R., VIEIRA, L. S.; DOMINGUES, M. B. P.; **GOMES, V. C. F.**. Hackathon - Introdução à Data Science. Applied Computing Workshop (WORCAP-2019). Short course, 2019. Available in <https://github.com/gqueiroz/worcap-hackathon>. **(I)**.

d) **GOMES, V. C. F.**; COSTA, R. W.; QUEIROZ, G. R.. Docker: introdução à administração de containers. Applied Computing Workshop (WORCAP-2018). Short course, 2018. Available in <https://github.com/vconrado/mc9-worcap2018>. **(I)**.

e) SIMOES, R.; SANCHEZ, A.; VINHAS, L.; QUEIROZ, G. R.; **GOMES, V. C. F.**. Python for Data Science in Earth Observation Analysis. CEOS - Working Group on Information Systems and Services (WGISS Tech Webinar). Webinar, 2017. Available in <https://github.com/rolfsimoes/wgiss-py-webinar>. **(I)**.

f) QUEIROZ, G. R.; SIMOES, R. E. O. ; **GOMES, V. C. F.**. Big Geospatial Data: Teoria e Prática com SciDB e Python. Short course, 2017. **(I)**.

1.3 Document structure

The structure of this thesis is based on the three papers presented in the following chapters. Chapter 2 is a manuscript published in the special issue "Spatial Data

Infrastructures for Big Geospatial Sensing Data" of the Remote Sensing journal. This paper presents the results of a study of the main platforms available for big EO data management and analysis. For each of the seven technologies studied, a section is dedicated to explaining their operation, architecture, and functionalities. This work also presents a comparative analysis of these platforms in relation to ten capabilities, which were defined based on the demands and needs presented by Camara et al. (2016) and Ariza-Porras et al. (2017). In addition to discussing the capabilities of each platform, this article provides a table and six graphs to summarize and illustrate this assessment. We also discussed the difficulty of a platform to provide a greater degree of abstraction simultaneously with flexibility in data-processing approaches. In this article, we suggest that a possible solution is to provide scientists with a platform with two forms of processing. The most frequently used features could be made available through a high abstraction API, similar to that provided by OpenEO and GEE. For more complex analyses, the platform could allow its extension through a framework, such as the solution adopted by the ODC, so that the scientist has direct access to data and infrastructure processing capabilities.

Chapter 3 is based on an article published in the "ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences". The objective of this study in the context of this thesis was to evaluate the possibility of using open tools integrated with the BDC platform's data and software products. For this purpose, this study presents an integration process between the data products of the BDC and ODC framework. This integration expands the services and tools that can be used to access, view, and analyze the EODC produced by the BDC project. In addition, this integration allows the algorithms previously developed for ODC technology to be more easily adapted for use with BDC data. In this study, a tool was developed and two ODC tools were adapted to work properly with the data produced by the BDC project. An architecture for interactive analysis is proposed and implemented, and we present the results of the processing performed on the BDC project infrastructure using ODC tools with BDC EO data.

Based on the knowledge acquired in these two previous works and on the demands of the BDC project, an architecture is proposed to allow users to process BDC EO data in the BDC's on-premise servers. The objective of this work is to provide a structure that allows the distributed processing of EO data, especially those recurrently executed on the BDC platform and in other INPE projects. To provide greater flexibility for data processing, the architecture was structured in the form of a framework that allowed the inclusion of new algorithms and the extension of previously existing data models. To provide a greater degree of abstraction, the proposed architecture provides an OpenEO API that allows users to access data and trigger server-side processing. The description of this architecture, the implementation of a prototype, and the results of two test cases are presented in Chapter 4, entitled "Brazil Data Cube Workflow Engine: a tool for big Earth Observation data processing" which will be used as a basis for the preparation of an article with the same title.

Chapter 5 presents the final remarks and future works.

2 AN OVERVIEW OF PLATFORMS FOR BIG EARTH OBSERVATION DATA MANAGEMENT AND ANALYSIS[1]

Nowadays, Earth observation (EO) data plays a crucial role in understanding processes on our planet, enabling us to make great strides in monitoring environmental change, risk detection and urban occupation analysis (APPEL et al., 2018; STROMANN et al., 2020). Based on information extracted from EO data, researchers and decision-makers are able to conceive and apply effective policies for environment protection and sustainable management of natural resources.

In recent years, the amount of EO data freely available for society and researchers has quite increased, driven by technological advances and open data policies adopted by governments and space agencies. Only in 2019, the volume of open data produced by Landsat-7 and Landsat-8, MODIS (Terra and Aqua units), and the three first Sentinel missions (Sentinel-1, -2 and -3) is around 5 PB (SOILLE et al., 2018). These big data sets often exceed the memory, storage, and processing capacities of personal computers; imposing severe limits that lead users to take advantage of only a small portion of the available data for scientific research and operational applications (MÜLLER et al., 2010; CAMARA et al., 2016; STROMANN et al., 2020). Thus, there is a need for novel technological solutions to properly store, process, disseminate, and analyze these big EO data sets.

A Spatial Data Infrastructure (SDI) is a platform that facilitates the interface between people and systems in the exchanging and sharing of spatial data, including EO information, by providing required technologies, policies, and standards (RAJABIFARD; WILLIAMSON, 2001). SDI as a sharing platform aims to facilitate the access and integration of multi-source spatial data inside a framework with a number of technological components. In the last years, traditional SDIs have been built based on standards to represent and store spatial information in data files (e.g. GeoTIFF and Esri Shapefile) and database systems (e.g. PostGIS and Oracle Spatial), as well as to serve spatial data, metadata and processes via web services, such as Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS), Catalog Service for Web (CSW) and Web Coverage Processing Service (WCPS). These standards proposed by Organization for Standardization (ISO) and the Open Geospatial Consortium (OGC) (OPEN GEOSPATIAL CONSORTIUM (OGC), 2019) are essential to enable the spatial data interoperability among SDIs

[1]This chapter is based on a paper published in the special issue "Spatial Data Infrastructures for Big Geospatial Sensing Data" of the journal Remote Sensing (GOMES et al., 2020). The paper is authored by Vitor C. F. Gomes, Gilberto R. Queiroz, and Karine R. Ferreira

from worldwide agencies and institutions. Although there are ISO and OGC web services that allow data processing in the server side such as WCPS, most current SDIs on regional, national, and international levels are focused on EO data sharing and dissemination in the form of individual files, through HTTP, FTP and SSH protocols, web services for data access, and web portals (MÜLLER, 2016).

In the current era of big spatial and EO data, there is a need for a next generation of SDIs able to properly deal with this vast amount of data (YUE et al., 2015; YUE et al., 2016; WOODCOCK et al., 2016; SEDONA et al., 2019). To take advantage of all potential of this era, users need to handle hundreds (or thousands) of EO data files, of different spectral, temporal, and spatial resolutions, by using or developing software scripts to extract information of interest. To meet these demands, novel technologies have been proposed and developed, based on cloud computing and distributed systems.

According to Merticariu et al. (2015), Array Database Management Systems (Array DBMS) have set out to fill an important gap in storage and management of EO data represented as regular and multidimensional arrays, such as satellite images. Unlike Relational Database Management Systems (RDBMS), which are based on the concept of *relations* (or *tables*) and relational algebra, Array DBMS are centered on multidimensional arrays. Examples of Array DBMS are RasDaMan (BAUMANN et al., 1998), SciDB (STONEBRAKER et al., 2013) and TileDB (PAPADOPOULOS et al., 2016). Most array DBMS have the capability to split large arrays into indexed blocks or chunks that are stored and shared among multiple computers to improve efficiency and performance. Besides that, they provide their own query language for multidimensional arrays, such as RasDaMan RasQL, SciDB AFL (Array Functional Language), and AQL (Array Query Language).

Another approach that has been explored for managing and handle big EO data is MapReduce systems. Traditionally, high-performance computing systems separate compute nodes and storage nodes, which are interconnected with high-speed links to meet data access requirements. However, the capacity of these high-speed links is still less than the aggregate bandwidth of all compute nodes (GUO et al., 2012). In parallel data systems such as Google File System (GFS) (GHEMAWAT et al., 2003) and Hadoop Distributed File System (HDFS) (SHVACHKO et al., 2010), clusters are built from standard servers and each node takes on both compute and storage functions. In these systems, data location stands out as a significant advantage over traditional systems by leveraging data location and reducing network traffic

- one of the bottlenecks in computing applications that handle a large amount of data (GUO et al., 2012; BLOMER, 2014; WANG; YING, 2016). Assis et al. (2017) propose an approach to use MapReduce systems for EO satellite image time series analysis. A comparison between Hadoop and SciDB in processing EO satellite images is presented by Camara et al. (2016).

Despite these novel technologies are addressing issues of storage and access, an increasing number of applications have reached data volumes and acquisition speeds that make it impossible to move data to local systems for processing and analysis (WOODCOCK et al., 2016; WU et al., 2018). The Moving Code paradigm stands out as an alternative in this scenario. In this approach, the executable application (or source code) is sent to run on the servers where the data is located, avoiding data movement (MÜLLER, 2016). To properly deal with big EO data, the next generation of SDIs should provide solutions based on the Moving Code paradigm and the approach to process data on the server side where data is stored.

Regarding web services protocols that allow EO data processing on the server side, we can cite two examples WCPS proposed by OGC and Web Time Series Service (WTSS). WCPS is an extension of WCS which allows server-side coverage manipulation, extraction, and analysis through a functional language (BAUMANN, 2010). This language allows the retrieving of one or more server-side coverages and performing of operations on the data. The EarthServer project has a WCPS implementation using the RasDaMan array database as a backend (BAUMANN et al., 2016). WTSS is a lightweight web service for time series recovery from big amounts of remote sensing images organized as coverages (VINHAS et al., 2016).

Recently, a solution that has facilitated the building of infrastructures based on Moving Code paradigm is the cloud computing environment offered by providers as services, such as Amazon Web Services (AWS) and Microsoft Azure Cloud Services. These environments often are highly scalable and provide developer tools, storage, database, and networking services. Besides providing services for virtual machines and containers, AWS contains several of the most used EO satellite image collections as open data, including Landsat-8, Sentinel-2, and CBERS-4 (Open Data on AWS (AMAZON WEB SERVICES, 2020)). Following this trend, the European Commission has financed five systems for cloud computing in the context of DIAS (Data and Information Access Service) (COPERNICUS, 2020): CREODIAS (CREODIAS, 2020), Mundi (MUNDI WEB SERVICES, 2020), ONDA (ONDA, 2020), WEkEO (WEKEO, 2020), and Sobloo (SOBLOO, 2020). These systems provide open EO data from Copernicus program through standard web services and protocols such as OpenSearch, WFS, WMS, WCS, and CSW (CREODIAS, 2020; MUNDI WEB SERVICES, 2020; ONDA, 2020; WEKEO, 2020; SOBLOO, 2020). Besides that, these systems allow users to contract on-demand services to store and process EO data sets as well as to directly access these data sets. Similar to AWS, the DIAS systems allow users to execute their applications in a cloud environment where data is stored, avoiding the movement of data through the network and so improving the processing performance.

Although the cloud computing environments as AWS and DIAS facilitate the high scale processing of EO data, considerable effort and advanced technical knowledge are still required for users to take advantage of these cloud computing environments. In addition to analytical tasks, the EO community scientists and users face with challenges to the management of these environment computational resources. Thus, there is a need for user-friendly solutions that provide high-level analytical tools for EO community researchers and users, abstracting the technical issues of these cloud computing environments. The Sentinel Hub platform is an example of this kind of solution (SINERGISE, 2020a). This platform uses the CREODIAS cloud computing services to access and process the Copernicus EO data sets and provides a high-level programming interface that allows users to deal with these data sets without worrying about technical details.

In this context, we define *"Platforms for big EO Data Management and Analysis"* as computational solutions that provide functionalities for big EO data management, storage and access; that allow the processing on the server side without having to download big amounts of EO data sets; and that provide a certain level of data and processing abstractions for EO community users and researchers. These platforms integrate different kinds of technologies, Application Programming Interfaces (API), and web services to provide a more complete solution for big EO data management and analysis. Based on this definition, this Chapter presents an overview of seven platforms and a comparison among their functionalities: Google Earth Engine (GEE) (GORELICK et al., 2017), Sentinel Hub (SH) (SINERGISE, 2020a), Open Data Cube (ODC) (OPEN DATA CUBE, 2022b), System for Earth Observation Data Access, Processing and Analysis for Land Monitoring (SEPAL) (FOOD AND AGRICULTURE ORGANIZATION (FAO), 2020), OpenEO (PEBESMA et al., 2017), JEODPP (SOILLE et al., 2018), and pipsCloud (WANG et al., 2018). These platforms have close purposes but use different storage systems, access interfaces, and abstractions for EO data sets.

This Chapter is organized as follows. In Section 2.1, we present an overview of the platforms where their main functionalities and technical issues are described. In Section 2.2, ten capabilities of the EO community interest are presented and used to compare the platforms presented. Finally, in Section 2.3, we discuss the platforms that stand out in the evaluated capabilities, the challenges in building a platform that fully meets the evaluated capabilities, and the new trends in these technologies.

2.1 Platforms for big Earth observation data management and analysis

This section presents the features, available functionality, and technical details of the seven platforms evaluated in this paper. The questions that are addressed about each platform in this section are: i) who supports it?; ii) how is it made available?; iii) what is the software architecture?; iv) how is the data stored?; v) how is the data processed?; vi) how is the data made available?; vii) what are the data abstractions used?; viii) is it possible to extend the platform?; and ix) is there support for reproducible science?

2.1.1 Google Earth Engine

Google Earth Engine (GEE) is a cloud-based platform that enables large-scale scientific analysis and visualization of geospatial data sets. GEE was launched in 2010 by Google as a proprietary system. Currently, it is available to users as a free service for small and medium workloads, using a business model similar to the other cloud-based services of the company.

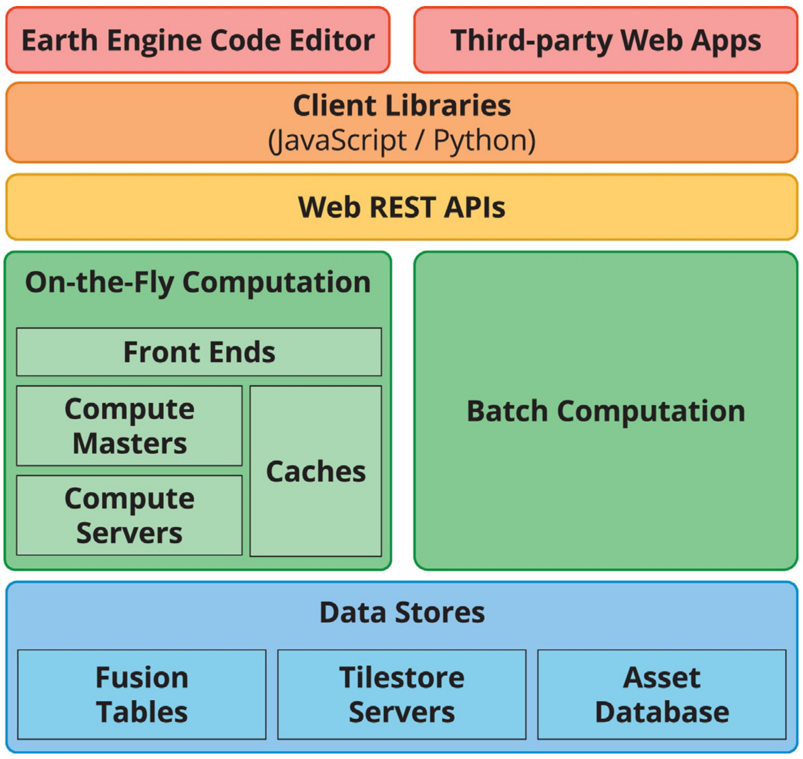
This platform is built from a collection of technologies available on Google's infrastructure, such as the large-scale computer cluster management system (Borg), the distributed databases (Bigtable and Spanner), the distributed file system (Colossus), and the parallel pipeline execution framework FlumeJava, as depicted in Figure 2.1 (GORELICK et al., 2017).

GEE provides a JavaScript API and a Python API for data management and analysis. For the JavaScript version, a web Integrated Development Environment (IDE)[2] is also provided, where the user has easy access to available data, applications and real-time visualization of the processing results. The Python API is available through a module and has a structure similar to its JavaScript version.

This platform provides a data catalog that stores a large repository of geospatial

[2]<https://code.earthengine.google.com>

Figure 2.1 - A simplified Google Earth Engine system architecture diagram.



SOURCE: Gorelick et al. (2017).

data, including optical imagery of a variety of satellites and air systems, environmental variables, weather and climate forecasts, land cover, socioeconomic, and topographic datasets. Before being made available, these data sets are preprocessed, enabling efficient access and removing many barriers associated with data management (GORELICK et al., 2017).

GEE uses four object types to represent data that can be manipulated by its API. The *Image* type represents raster data that can consist of one or more bands, which contain a name, data type, scale, and projection. A stack or a time series of *Images* is represented by the *ImageCollection* type. GEE represents vector data through the *Feature* type. This type is represented by a geometry (point, line, or polygon) and a list of attributes. The *FeatureCollection* type represents groups of related *Features* and provides functions to manipulate this data, such as sorting, filtering, and visualization.

To process and analyze data available in the GEE public catalog or data from the user's private repository, GEE provides an operator library for the object types listed above. These operators are implemented in a parallel processing system that automatically splits the computation so that it can be performed in a distributed environment.

In the GEE approach, objects handled via JavaScript or Python represent proxies for data stored in Google's infrastructure. When a new object is generated as a result of some operation, the locally stored object has the description of the operation and the addresses of the input objects of the operation. Processing only occurs when there is an output (visualization or writing) of the object produced for computation. This lazy computation mode allows the processing of only some parts of the data that actually produce the required output.

The result of a GEE processing can be viewed in the web IDE or saved in one of three company services: *Drive*, *Cloud Storage* or *Assets*. GEE uses a *Tiles* server to make data available to the web interface efficiently. However, this service is not explicitly provided and so users can not integrate it into other applications.

As a computing system with shared resources among users, GEE imposes limits on the iterative usage mode (*On-the-Fly Computation* box in Figure 2.1), where execution takes place in real-time. The limits are on the maximum duration of each request, the maximum number of simultaneous requests per user, and the maximum execution of operations considered costly, such as spatial aggregations. Processing beyond thresholds is possible through batch computing mode (*Batch Computation* box in Figure 2.1) (GORELICK et al., 2017). For data-intensive processing or compute-intensive analytics, there is currently a need to make a request to Google.

The Google Earth Engine library has over 800 functions to handle big EO data sets. Despite this large number of builtin functions, these functions are close and users can not update or extend their basic functionalities. While GEE provides a friendly environment for scientists, the implementation of procedures that are not available through the GEE API functions requires significant user effort (SHELESTOV et al., 2017). Besides that, GEE only offers programming interfaces that support pixel-based processing, and there are natively no region-based methods such as image segmentation or large-scale time series analysis.

The characteristics of GEE have motivated its use by the EO community researchers. There are, for example, studies using this platform for urban area assessment (GOLDBLATT et al., 2016) and evaluations of the performance of algorithms available in the platform for the classification of agricultural crops (SHELESTOV et al., 2017).

One facility that GEE provides to users is the capability to share their scripts and assets with other users of the platform. Nevertheless, it is important to keep in mind that these scripts use algorithms implemented internally by the platform and that these algorithms are close and can not be extended on the server side. Therefore, GEE can not guarantee that an analysis performed on the platform can be reproduced in the future, since these internal algorithms can be changed or discontinued by the platform at any time (GOOGLE, 2020).

The GEE terms of service guarantee users the intellectual property of their codes and data as well as that the company does not use such information for purposes other than what is necessary to provide the service to the user (GOOGLE, 2020).

2.1.2 Sentinel Hub

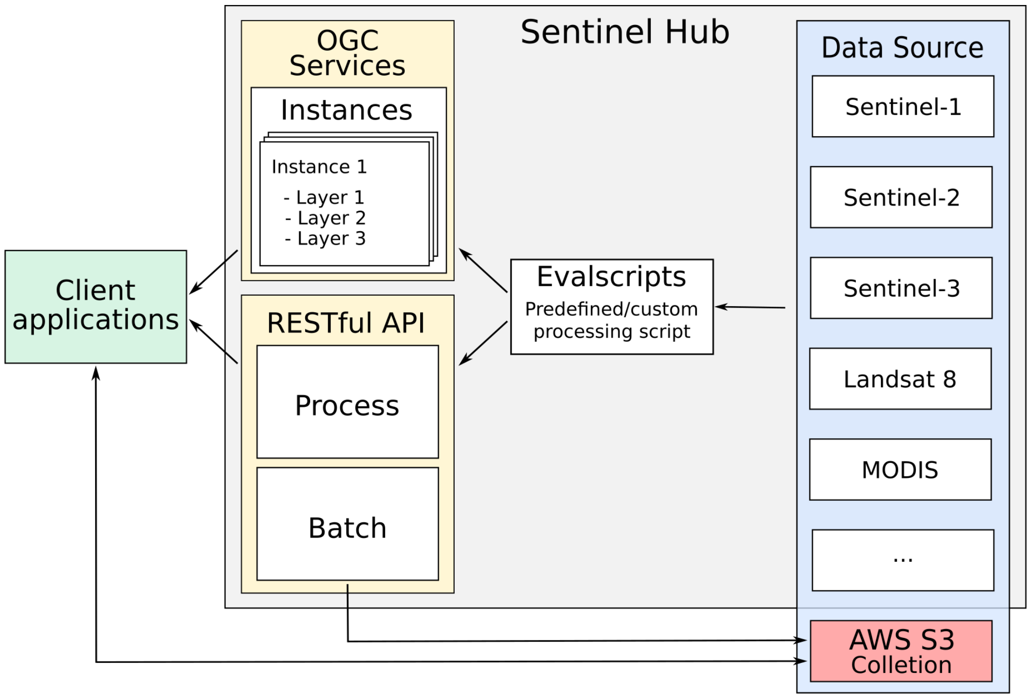
Sentinel Hub (SH) is a platform developed by Sinergise that provides Sentinel data access and visualization services. This is a private platform with public access[3]. Unlike Google Earth Engine, SH limits access to functionality in different payment plans. The free plan only allows viewing, selection, and downloading of raw data. Paid access enables data access through OGC protocols and a specific API, data processing, mobile application data access, higher resource access limits, and technical support (SINERGISE, 2020a).

The features of the SH platform are made available through OGC services and a RESTful API. A web interface is also available that allows the configuration of specific services. Sinergise does not provide a diagram of the system architecture used by Sentinel Hub or information about how data is stored or processed. Thus, Figure 2.2, prepared by us, intends to illustrate the interaction between the services provided and the data abstractions used by the SH platform. In this figure, the arrows indicate the direction of the data flow.

SH uses the concepts of *Data Source*, *Instances*, and *Layer* to represent the data available in its services. *Data Source* is an abstraction equivalent to the GEE *ImageCollection*, representing datasets that share the same set of bands and metadata.

[3]<https://www.sentinel-hub.com>

Figure 2.2 - Sentinel Hub services and data abstraction diagram.



The *Data Sources* currently available on SH are: Landsat 8 L1C, Mapzen DEM, MODIS MCD43A4, Sentinel-1 GRD, Sentinel-2 L1C and L2A, Sentinel-3 OLCI and SLSTR, and Sentinel-5P L2. It is also possible to include your own data in the SH platform. These data sets are named *Collections* by SH and need to be stored in the cloud optimized GeoTIFF (COG) format in a S3 bucket.

An *Instance* in SH platform acts as a distinct OGC service that can be configured to provide a set of *Layers* that fulfill user needs. Each *Layer* is associated with one or more bands of a specific *Data Source* and a processing script. These scripts, called *Evalscripts* by the SH, are applied to each pixel of the data requested by the user. It is not possible to access the data of a pixel's neighborhood during the execution of the script, which can basically perform operations between bands.

Evalscripts are also used for SH API Process and Batch modules. In the Process module, the user can request a *Data Source*, informing filtering parameters, such as bounding box and time range, and indicating an *Evalscript* that will be applied to each pixel of the data before being delivered by the service. In the Batch mode the behavior is the same, except the service that is asynchronous and saves the result in an S3 bucket informed by the user during the request (SINERGISE, 2020b). The SH web interface provides an interactive means for creating *Instances* and registering *Collections* by users.

The Sentinel Hub source code is close and so it is not possible to extend it. Sinergise provides good documentation with examples for using the web interface and SH API. Similar to GEE, Sentinel Hub terms of service guarantee users the intellectual property of their content and that the company will use it only for the provision of the platform services. Also, Sentinel Hub can change the offered functionalities at any time and there is no guarantee of continuity of services (SINERGISE, 2020a). Currently, SH has no tools to facilitate the sharing of *Evalscripts* among researchers.

2.1.3 Open Data Cube

The Open Data Cube (ODC), previously known as the Australian Geoscience Data Cube (AGDC), is an analytical framework composed of a series of data structures and tools that facilitate the organization and analysis of EO data. It is available under Apache 2.0 license as a suite of applications. It is currently supported by Analytical Mechanics Associates (AMA), The Committee on Earth Observation Satellites (CEOS), The Commonwealth Scientific and Industrial Research Organization (CSIRO), Geoscience Australia (GA), and United States Geological Survey (USGS).

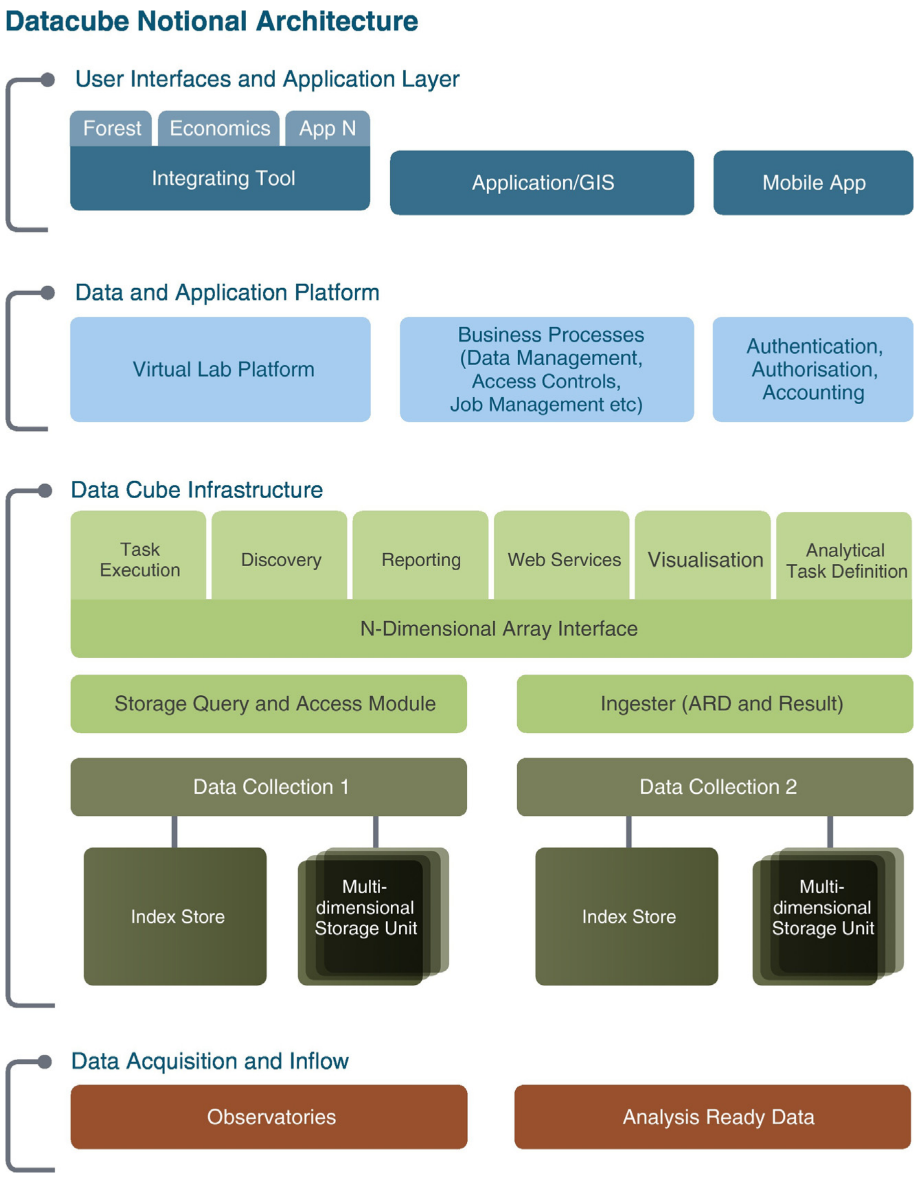
ODC allows the cataloging of massive EO data sets and their access and manipulation through a set of command line tools and a Python API. Figure 2.3 illustrates the architecture of ODC. *Data Acquisition and Inflow* represents the process to collect and prepare EO data before indexed by ODC. *Data Cube Infrastructure* illustrates the main core of ODC, where EO data is indexed, stored and delivered to users over the Python API. *Data and Application Platform* group auxiliary application modules as job management and authentication.

The source code of ODC and its tools are open and are officially distributed through dozens of git repositories[4]. These repositories include web interface modules for data visualization, data statistics extraction tools as well as jupyter notebooks with examples of access and use of indexed data in ODC.

The main module responsible for data indexing is called datacube-core and consists of a Python script set that uses a PostgreSQL database to catalog the metadata of

[4]<https://github.com/opendatacube>

Figure 2.3 - Architecture diagram of the Open Data Cube platform.


Datacube Notional Architecture
User Interfaces and Application Layer
Forest
Economics
App N
Integrating Tool
Application/GIS
Mobile App
Data and Application Platform
Virtual Lab Platform
Business Processes (Data Management, Access Controls, Job Management etc)
Authentication, Authorisation, Accounting
Data Cube Infrastructure
Task Execution
Discovery
Reporting
Web Services
Visualisation
Analytical Task Definition
N-Dimensional Array Interface
Storage Query and Access Module
Ingester (ARD and Result)
Data Collection 1
Data Collection 2
Index Store
Multi-dimensional Storage Unit
Index Store
Multi-dimensional Storage Unit
Data Acquisition and Inflow
Observatories
Analysis Ready Data

SOURCE: Lewis et al. (2017).

the data and provides an API for data retrieval. ODC can index data stored on a file system or the web. This platform does not use any approach to data distribution between servers, and it is the user's chosen file system's responsibility to ensure the scalability of data storage and access.

ODC uses the concepts of *Product* and *Dataset* to represent the data indexed in its catalog. *Products* are collections of data sets that share the same set of measures and some subsets of metadata. The *Dataset* represents the smallest independently described, inventoried, and managed data aggregation. These are usually scenes stored in files that together represent a *Product* (OPEN DATA CUBE, 2021).

The data loading process in ODC consists of four steps. First, a *Product* must be registered from its metadata. In the second step, the metadata is extracted from each file (*Dataset*) that will be linked to the *Product*. To automate this process, ODC provides metadata extraction scripts for the following instruments/sensors: Landsat-5/7/8, Sentinel-1/2, ALOS-1/2, ASTER Digital Elevation Model (DEM), and MODIS (OPEN DATA CUBE, 2022b). With the metadata of the prepared files, the third step is to register these *Datasets* in the ODC catalog. The last step, called data ingestion, is optional and deals with the process of creating new *Datasets* from already registered *Datasets*. These new *Datasets* are saved using a new storage scheme, composing a new *Product*. At this stage, data can be re-sampled, reorganized in time series, and/or broken into smaller blocks for storage. The main purpose of producing these new *Datasets* is to optimize the data format to speed up the reading process of data in the file system.

The development of applications for processing data indexed by an ODC instance requires the use of the framework's Python API. This API allows listing of indexed *Products*, retrieving *Products* and *Datasets* data and metadata. Data retrieval is performed by a `load` function that receives parameters such as product name, bounding box, period range, and spatial output resolution. This function returns a `xarray` object to be used by user's application. For parallelization of processing, ODC has examples of using asynchronous task queues through the use of the Celery framework. Nevertheless, the parallelization of the processing of applications using ODC is the responsibility of the user.

For data access, ODC provides implementations of OGC web services such as WCS, WMS and WMTS. In addition to these modules, ODC also provides tools that facilitate ODC deployment through containers, an implementation that exposes the ODC Python API through a REST API, Jupyter Notebooks with Sample Applications, an application for extracting statistics from indexed data and an implementation of a web portal. In this web portal, it is possible to discover the available products, select and retrieve datasets (GeoTIFF and NetCDF), and perform analyzes through applications available in the portal.

Out of the box, this web portal offers a small set of sample applications, but new features can be added. This requires using a template provided by ODC developers.

In this template, the researcher needs to extend classes from the Django framework and the Celery framework (OPEN DATA CUBE, 2020).

Currently the ODC platform has no tools to facilitate the sharing of applications and data between researchers. For a user to be able to reproduce results in another instance of ODC it is necessary to manually share and index the data and applications used.

In June 2019, 9 institutions were using ODC operationally, 14 were in the implementation phase and 33 were analyzing its implementation (OPEN DATA CUBE, 2019). The most prominent instance is the Australian version, called the Australian Geoscience Data Cube (AGDC). This instance stores over 300,000 Landsat images covering Australia, corresponding to approximately 10^{13} measurements stored. To store this volume of data, AGDC uses a cluster of computers with the Lustre distributed file system (LEWIS et al., 2017). Two other operational instances are Switzerland (GIULIANI et al., 2017) and Colombia (ARIZA-PORRAS et al., 2017).

2.1.4 SEPAL

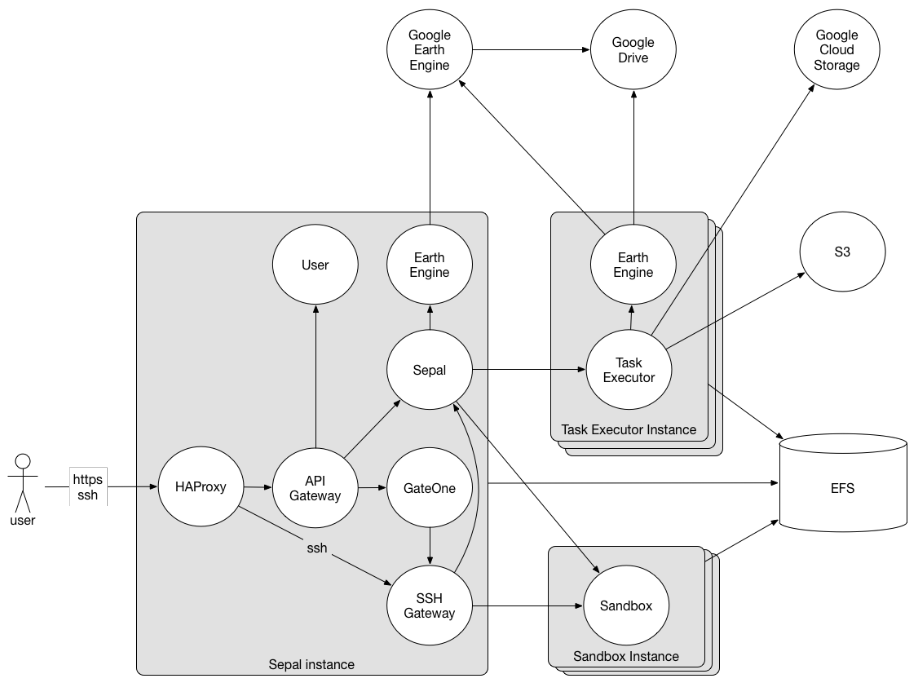
The System for Earth Observation Data Access, Processing and Analysis for Land Monitoring (SEPAL) is a cloud computing platform developed for the automatic monitoring of land cover. It combines cloud services, such as Google Earth Engine, Amazon Web Services Cloud (AWS), with free software, such as Orfeo Toolbox, GDAL, RStudio, R Shiny Server, SNAP Toolkit, and OpenForis Geospatial. The main focus of this platform is on building an environment with previously configured tools and on managing the use of computational resources in the cloud to facilitate the way scientists search, access, process, and analyze EO data, especially in countries that have difficulties with connection with the Internet and few computational resources (FOOD AND AGRICULTURE ORGANIZATION (FAO), 2020).

SEPAL is an initiative of the Forestry Department of the United Nations Food and Agriculture Organization (FAO) and financed by Norway. Its source code[5] is available under MIT license and is still under development (FOOD AND AGRICULTURE ORGANIZATION (FAO), 2020). It works as an interface that facilitates access and integration of other services. Figure 2.4 presents a diagram of SEPAL architecture. The *Sepal Instance*, provides a web-based user interface where users can search and retrieve datasets and start preconfigured cloud-based machines to perform their analysis.

[5] <https://github.com/openforis/sepal>

SEPAL uses *Google Cloud Storage* and *Google Drive* to store EO data and metadata. The Google Earth Engine is used for data processing and retrieval and the Amazon Web Services Cloud (AWS) is used for data storage (S3 and EFS AWS services) and as an infrastructure for computing analyses (EC2 AWS service).

Figure 2.4 - Architecture diagram of the SEPAL platform.



SOURCE: FOOD AND AGRICULTURE ORGANIZATION (FAO) (2020).

The SEPAL platform can be accessed through a web portal[6], running on the AWS infrastructure, or it can be installed on the user own infrastructure using Vagrant for the management of processing instances. Currently, there are few available documentations about the deployment on in-house infrastructure.

In the web portal, the functionalities are divided into 4 areas: *Process*, *Files*, *Terminal*, and *Apps*. In *Process*, the user can search and retrieve images for further processing or viewing, by selecting the area, the sensor (Landsat or Sentinel-1 and 2) and the period of interest on the web interface. After searching, it is possible to select the best scenes for the production of a mosaic that can be downloaded to a user's storage space. In the *Files* section, users can browse through the files previ-

6<https://sepal.io>

ously saved in their storage space. The files searched and retrieved in the *Search* section can be accessed and viewed from this area of the portal. The area called *Terminal* allows users to start a machine in the AWS cloud. Before executing it, users must choose one of the 22 hardware configurations available. Each machine is associated with a cost per hour of use. Upon accessing the SEPAL account, users receive a fixed amount of credits monthly. The machine configuration options available are some possibilities found in the AWS EC2 service. In the *Apps* area, applications are available to process and analyze data previously stored in the user's storage space. When an option is selected, SEPAL deploys the application code to the user's running machine (or instantiates a new one, if none is active) and opens a new browser window pointing to its interface. Currently available applications are RStudio, Jupyter Notebook, Jupyter Lab, and interactive documents that run on an R Shiny Server.

Although SEPAL provides functionality for users to manage and view data on its web interface (*Files*), this platform does not provide any web service to access the data or send processing requests to the server. Its features are more focused on the management of computational resources (virtual machines) and the aid of data pre-processing from EO data providers through a web interface (*Process*). SEPAL automatically connects the storage service to the virtual machines allocated by the users, making the previously downloaded files available to the user's applications. In this environment, researchers are responsible for developing applications that make good use of available computational resources. SEPAL also does not provide tools to facilitate the sharing of analysis between users.

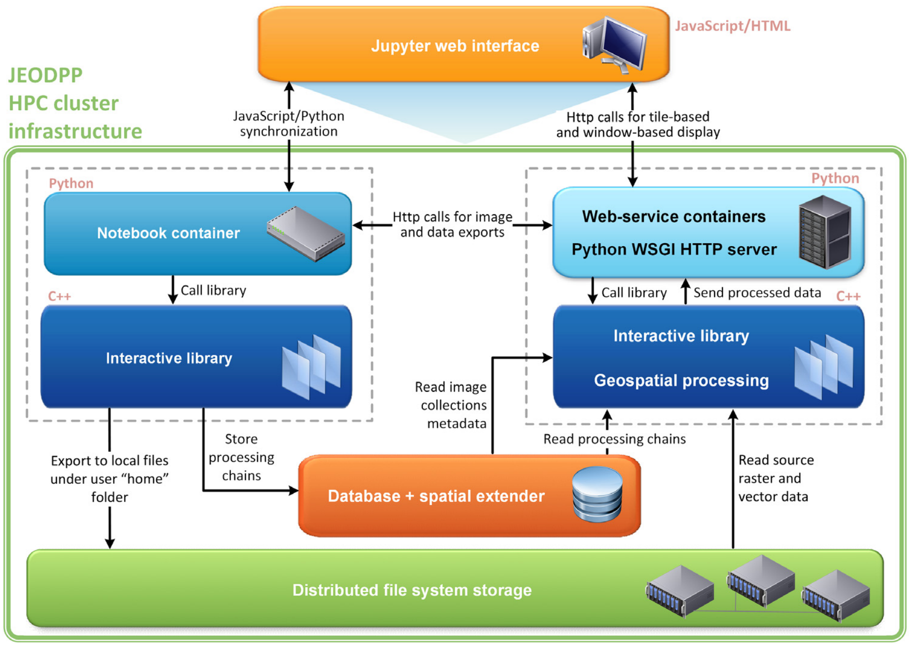
2.1.5 JEODPP

The JRC Earth Observation Data and Processing Platform (JEODPP) is a closed solution developed since 2016 by the Joint Research Center (JRC) for the storage and processing of large volumes of Earth observation data. This platform has features for interactive data processing and visualization, virtual desktop, and batch data processing. This platform uses a set of servers for data storage and another set for processing. The storage servers use the EOS (ADDE et al., 2015) distributed file system and store the data in its original format, with only pyramidal representations added to speed up the reading and visualization of the data.

For data visualization, JEODPP uses a Jupyter Notebook environment and provides an API for the construction of objects that represent processing chains, through previously defined functions. When a visualization object is built, the associated processing chain is not executed instantly. The execution of the processing chain only occurs when data associated with the object is used. This lazzy processing approach is the same as that adopted by GEE for data visualization.

Figure 2.5 illustrates the data processing and visualization flows adopted by JEODPP. The left side shows the flow of registration of the processing chains, while the right side shows the flow of data delivery for visualization.

Figure 2.5 - System architecture for interactive visualization and processing of EO data on JEODPP platform.


Jupyter web interface
JavaScript/HTML
JEODPP HPC cluster infrastructure
JavaScript/Python synchronization
Http calls for tile-based and window-based display
Python
Notebook container
Http calls for image and data exports
Python
Web-service containers
Python WSGI HTTP server
Call library
C++
Interactive library
Call library
Send processed data
C++
Interactive library
Geospatial processing
Read image collections metadata
Read processing chains
Export to local files under user "home" folder
Store processing chains
Database + spatial extender
Read source raster and vector data
Distributed file system storage

SOURCE: Soille et al. (2018).

For the virtual desktop feature, JEODPP uses the Apache Guacamole system, which allows viewing a remote terminal through a browser. This feature allows the use of remote Linux or Windows terminals, which are previously prepared with tools for processing EO data (R, Grass, GIS, QGIS, MATLAB, etc.) and where the data is available through the EOS file system.

For batch processing, JEODPP uses the HTCondor framework for scheduling tasks on servers. Users are responsible for integrating HTCondor into their applications in order to take advantage of the cluster of processing servers. In these processing modes, users directly access the files, making it necessary to know the folder structure and the formats of the stored data. JEODPP does not provide any kind of extra abstraction for accessing and manipulating this data.

Soille et al. (2018) presents the results of two tests performed with JRC applications in an instance of the JEODPP system running in a cluster with 912 cores (15.8TFLOS) and accessing 1PB of data. The tested applications are: i) Search for Unidentified Maritime Objects (SUMO); and ii) Global Human Settlement Layer (GHSL) framework. For each application, the number of pixels processed per second in Mpixel/s was calculated. The results show a linear growth of performance in relation to the increase of processor numbers, indicating good processing and data access scalability.

Based on the documentation available, the JEODPP does not have tools to facilitate the sharing of analysis among researchers. This capability is only available for internal use of JRC and there is no source code available for implementation in other institutions.

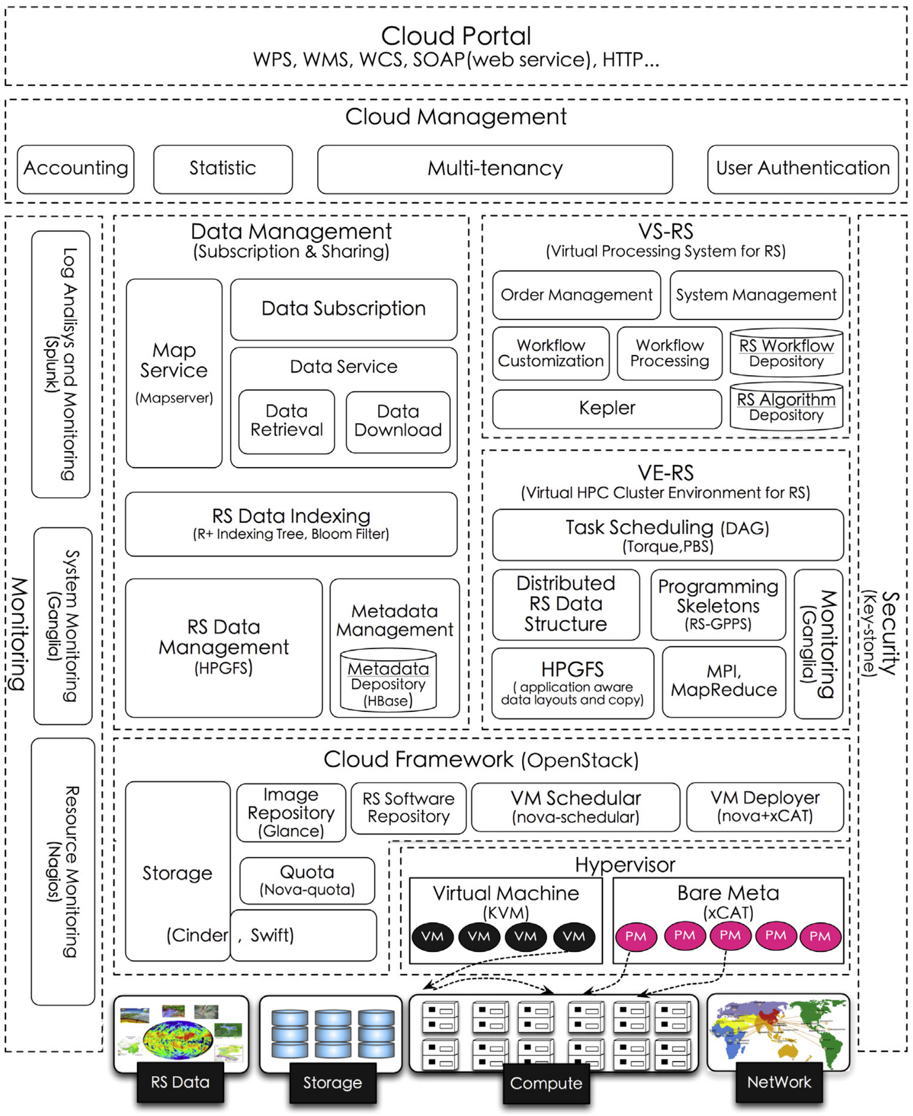
2.1.6 pipsCloud

PipsCloud (WANG et al., 2018) is a proprietary solution developed by Chinese research institutions for the management and processing of large volumes of EO data based on cloud computing. The file system used in pipsCloud is HPGFS, a proprietary file system also developed by Chinese institutions and which is not available for use by third parties. Its cloud environment is implemented in the organization's internal infrastructure using OpenStack technology, which allows the construction of virtualized services infrastructure.

Figure 2.6 shows the architecture of pipsCloud. The highlight is given to the file indexing scheme, which uses a Hilbert-R+ tree, and a virtual file catalog. Users who need to process data should describe a query to identify the files of interest. From these files, a file system is mounted on the user's processing machine with only the requested data.

For data processing on the server-side, pipsCloud provides C++ code templates for building applications. These templates abstract the reading of the data on disk and the submission of the processes to the processing nodes. The communication between the processing nodes and the parallel reading of the files is done using features of the MPI interface (Message Passing Interface).

Figure 2.6 - Architecture of pipsCloud platform.


Cloud Portal
WPS, WMS, WCS, SOAP(web service), HTTP...
Cloud Management
Accounting
Statistic
Multi-tenancy
User Authentication
Monitoring
Log Analisys and Monitoring (Splunk)
System Monitoring (Ganglia)
Resource Monitoring (Nagios)
Data Management (Subscription & Sharing)
Map Service (Mapserver)
Data Subscription
Data Service
Data Retrieval
Data Download
RS Data Indexing (R+ Indexing Tree, Bloom Filter)
RS Data Management (HPGFS)
Metadata Management
Metadata Depository (HBase)
VS-RS (Virtual Processing System for RS)
Order Management
System Management
Workflow Customization
Workflow Processing
RS Workflow Depository
Kepler
RS Algorithm Depository
VE-RS (Virtual HPC Cluster Environment for RS)
Task Scheduling (DAG) (Torque,PBS)
Distributed RS Data Structure
Programming Skeletons (RS-GPPS)
Monitoring (Ganglia)
HPGFS (application aware data layouts and copy)
MPI, MapReduce
Security (Key-stone)
Cloud Framework (OpenStack)
Storage (Cinder , Swift)
Image Repository (Glance)
RS Software Repository
VM Schedular (nova-schedular)
VM Deployer (nova+xCAT)
Quota (Nova-quota)
Hypervisor
Virtual Machine (KVM)
VM
Bare Meta (xCAT)
PM
RS Data
Storage
Compute
NetWork

SOURCE: Wang et al. (2018).

The pipscloud platform does not provide any functionality to export data using OGC standards or to facilitate the reproducibility of science. In addition, it is only available for internal use by the institutions that participate in the project and its source code is close and is not available for implementation in other institutions.

2.1.7 OpenEO

The OpenEO project started in October 2017 in order to meet the need to consolidate available technologies for storing, processing, and analyzing large volumes of EO data. This demand arises from the difficulty that many users of EO data have in migrating their data analytics to cloud-based processing platforms. The main reason is not, in many cases, of a technical nature, but the fear of becoming dependent on the provider of the chosen platform. OpenEO aims to reduce these concerns, by providing a mechanism for scientists to develop their applications and analyzes using a single standard that can be processed in different systems, even facilitating the comparison of these providers. With this approach, OpenEO aims to reduce the entry barriers for the EO community in cloud computing technologies and in big EO data analysis platforms.

To this end, this system has been developing as a common and open source[7] interface (Apache license 2.0) to facilitate the integration between storage systems and analysis of EO data and applications of the European program Copernicus.

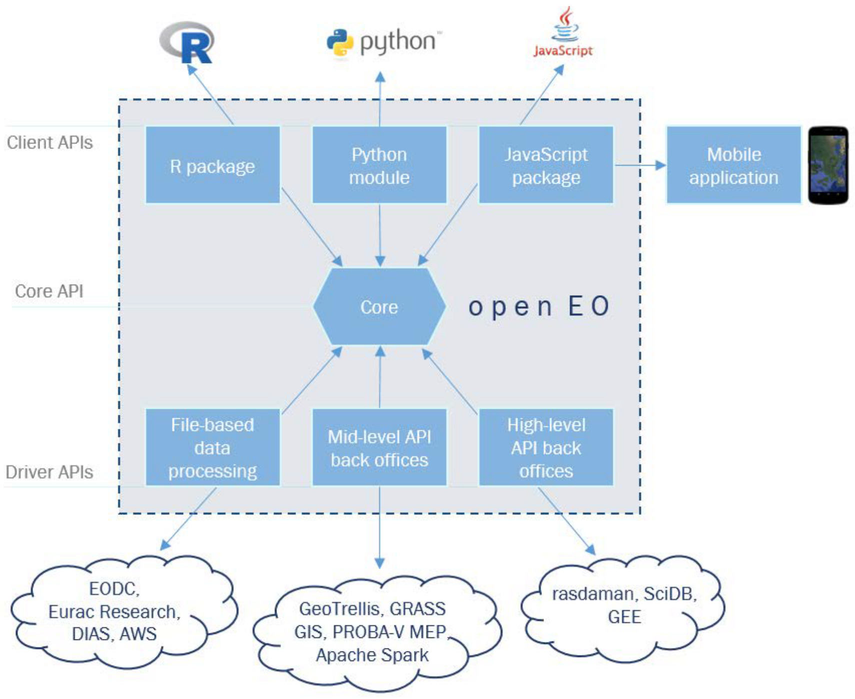
Figure 2.7 shows the three-tier architecture adopted by OpenEO. The *Client APIs* layer consists of packages or modules in R, Python, and JavaScript, which act as an entry point for the development of analyzes by researchers. This API uses two terms to describe EO datasets. The *Granule* refers to a limited area and represents the less granularity of data that can be managed independently. The *Collection* is a sequence of *Granules* sharing the same product specification (OPENEO, 2022). The *Core API* layer is responsible for standardizing *Client APIs* requests, unifying access to services provided by data processing technologies and platforms. Finally, the *Driver APIs* layer is responsible for the interface between the *Core API* and the data storage and processing services (back-end services).

OpenEO uses microservices to implement the *Core API*. These web services use REST architectural style and are divided into the following functionalities (OPENEO, 2018):

- *Capabilities*: retrieves the functionality of the back-end services, such as which authentication methods are supported and which *User-defined functions* (UDF) can be performed;
- *EO Data Discovery*: describes which data sets and image *Collections* are

[7]<https://github.com/Open-EO>

Figure 2.7 - Architecture of the OpenEO project.


python
JavaScript
Client APIs
R package
Python module
JavaScript package
Mobile application
Core API
Core
open EO
Driver APIs
File-based data processing
Mid-level API back offices
High-level API back offices
EODC, Eurac Research, DIAS, AWS
GeoTrellis, GRASS GIS, PROBA-V MEP, Apache Spark
rasdaman, SciDB, GEE

SOURCE: Pebesma et al. (2017).

available on back-end services;

- *Process Discovery*: provides the types of processing available at each back-end provider;
- *UDF Runtime Discovery*: allows the discovery of programming languages and environments for the execution of UDFs;
- *Job Management*: organizes and manages the tasks running at the back-end providers;
- *Result Access and Services*: provides the services for data recovery and processing results in the form of OGC Web Coverage Service (WCS) or OGC Web Map Service (WMS);
- *User Data Management*: manages the user account, such as storage and processing usage issues; and
- *Authentication*: provides user authentication.

The interaction between the *Client API* and *Core API* layers is made using 4 abstractions of processing elements, which are used to represent the objects sent by users to the OpenEO web services. The first, *Process Graphs*, defines process calls, including input parameters for previously defined functions. For more complex processing, these graphs allow the linking of multiple processes, where one process can receive another graph as a parameter, and so on. This type of computation invocation is equivalent to that used by GEE. *Tasks* represent another abstraction used in the processing flow adopted by OpenEO. These *Tasks* can be: Lazy evaluated jobs; Batch jobs; or Synchronously executed jobs. From the execution of a *Job*, it is possible to make its results available through OGC WCS or WMS.

The third abstraction used by OpenEO is *User Defined Functions*. They are used to expose, on the server side, data to applications in different ways. They represent the interface of the functions that users can implement to run on the server side. The last abstraction provided by OpenEO is *Data View*. This functionality allows the user to select and configure the temporal and spatial resolution of the data to be viewed. A *Data View* allows processing on demand, when only the data to be viewed is processed. This approach is similar to that adopted by the Lazy mode used by the GEE (OPENEO, 2018; OPENEO, 2022).

OpenEO does not restrict the technologies used in the back-end for data storage or processing. Thus, it is not possible to guarantee that all functionalities will be available or that the applications will work in the same way in different back-ends. Besides that, OpenEO does not provide facilities to guarantee the reproducibility of science.

2.2 Assessment of the platforms

We define *"Platforms for big EO Data Management and Analysis"* as computational solutions that provide functionalities for big EO data management, storage and access; that allow the processing on the server side without having to download big amounts of EO data sets; and that provide a certain level of data and processing abstractions for EO community users and researchers. EO community users need efficient and stable solutions that provide analytical tasks and that minimize their effort and required technological expertise to manage large EO data sets. These solutions should provide a certain level of data and processing abstraction that allows scientist to deal with large EO data sets without worrying about technologies such as database systems, web services and distributed computing. These technologies should be integrated and presented to scientists in the form of a platform where the complexities of data storage, processing, and infrastructure must be abstracted. According to Camara et al. (2016), an architecture for big EO data analysis must meet the following requirements:

- *Analytical Scalability*: it should cover the entire research cycle, allowing algorithms developed on personal computers to be executed in massive computing environments or large data volumes without major changes.
- *Software reuse*: it should allow researchers to adapt existing methods for application to large volumes of data with minimal rework.
- *Collaborative work*: it should allow results to be shared with the scientific community.
- *Replication*: it should encourage research groups to deploy their own infrastructure.

Ariza-Porras et al. (2017) list the following characteristics needed in a system for managing and analyzing EO data of interest to their research group:

- *Data ownership*: historical out-of-the-box data series should be available without reliance on external services.
- *Extensibility*: new processing algorithms should be easily added to the system.
- *Lineage*: the origin of the results should be identifiable by the algorithms and parameters used.
- *Replicability*: results should be replicable.
- *Complexity abstraction*: developers should be able to create new algorithms without interacting directly with the data query API. Developers should work with multidimensional arrays.
- *Ease of use*: The user interface should allow analysts to execute algorithms without long training. Developers should be able to create new algorithms using programming language knowledge with a small learning curve.
- *Parallelism*: Available computing resources should be used effectively by the system.

The demands and needs presented by Camara et al. (2016) and Ariza-Porras et al. (2017) motivated us to define ten capabilities to serve as criteria for evaluating the platforms presented in this work. The following list provides each capability name, its description, and, when applicable, its associated term proposed by other authors in parentheses:

- **Data abstraction**: the capacity to provide data abstraction, hiding from scientists details about how data is stored without limiting its access mode (*Complexity abstraction*; *Ease of use*).
- **Processing abstraction**: the capacity to provide data processing abstraction, hiding from scientists details about where and how data is processed, without limiting its processing power (*Complexity abstraction*).
- **Physical infrastructure abstraction**: the capacity to hide from scientists aspects regarding the number of servers, hardware and software resources (*Analytical scalability*; *Complexity abstraction*).
- **Open Governance**: the capacity of the scientific community to participate in the governance and development of the platform.
- **Reproducibility of science**: the capacity to provide means that allow scientist to share their analysis and/or reproduce the results among other researchers (*Collaborative work*; *Replicability*).
- **Infrastructure replicability**: the capacity to replicate the software stack, processes, and data on own infrastructure (*Replication*).
- **Processing scalability**: the capacity to scale processing performance by adding more resources (hardware/software) without a direct impact on the way scientists conduct their analysis (*Parallelism*).
- **Storage scalability**: the capability to scale storage space by adding more resources (hardware/software) without a direct impact on how scientists access data (*Parallelism*).
- **Data access interoperability**: the capacity to provide means, based on standardized interfaces, that allow other applications to access analysis results or data sets available in the platform.

- **Extensibility**: the capacity to add new software tools that utilize the storage and processing modules available internally in the platform (*Analytical Scalability*; *Software reuse*; *Extensibility*).

In this work, we evaluate these ten capabilities of the platforms Google Earth Engine (GEE), Sentinel Hub (SH), Open Data Cube (ODC), System for Earth Observation Data Access, Processing and Analysis for Land Monitoring (SEPAL), OpenEO, JEODPP, and pipsCloud. Table 2.1 presents a summary of this evaluation. For each platform, we classify each capability in three levels: *low*, *medium*, and *high*. The following paragraphs present details about the platform capabilities.

GEE, ODC, OpenEO, and SH provide high data abstraction to facilitate data access by users. On these platforms, users can make queries with spatial and temporal criteria to select the data set of interest. GEE uses the concept of *ImageCollection* to represent the data sets stored in these platforms. ODC uses the concept of *Product* and delivers data through a multidimensional data, while OpenEO uses the concept of *Collection* for these structures. SH provides access to data through *Data Sources* that can be accessed through services. SEPAL, JEODPP, and pipsCloud do not provide data abstraction. On these platforms, data is accessed through direct file manipulation.

Regarding the abstraction of data processing, we consider that no platform fully meets this capacity. GEE and SH platforms make transparent to users how and where processing is performed, but limit or hinder the use of a non-pixel-by-pixel processing approach. ODC does not have this limitation, but it requires the user to be aware of the processing mode used. In this platform, it is possible to perform sequential or parallel processing. For parallel mode, ODC provides a toolkit to facilitate application development, which should be developed by the user. OpenEO provides a complete interface for performing server-side processing transparently, but it demands that such functionality must be available on the back-end used. Thus, the processing abstraction may or may not be observed in the OpenEO, depending on the chosen back-end platform. SEPAL, JEODPP, and pipsCloud platforms do not have processing abstraction capabilities because they require users to develop their applications and access data directly through files. On these platforms, parallelization or distribution of processing must be implemented and managed by the users themselves.

GEE, OpenEO, and SH platforms provide a high abstraction of the physical infras-

Table 2.1 - Capacities of the platforms for big EO data management and analysis.

Capability	ODC	GEE	SEPAL	JEODPP	pipsCloud	OpenEO	SH
Data abstraction	**High**: Product and Dataset	**High**: Image, ImageCollection, Feature, and FeatureCollection	**Low**: Direct file handling	**Low**: Direct file handling	**Low**: Direct file handling	**High**: Collection and Granule	**High**: Data Source, Instances and Layers
Processing abstraction	**Medium**: Xarray and celery	**Medium**: Predefined pixel-wise functions	**Low**: User runs his own code	**Low**: User runs his own code	**Low**: User runs his own code	**Medium**: User-Defined Functions, Process graphs, and Jobs	**Medium**: Custom scripts (Evalscripts) layers perform pixel-wise processing
Physical infrastructure abstraction	**Medium**: Only data storage infrastructure	**High**: Both data storage and processing infrastructure	**Medium**: Only data storage infrastructure	**Medium**: Only data storage infrastructure	**Medium**: Only data storage infrastructure	**High**: Both data storage and processing infrastructure	**High**: Both data storage and processing infrastructure
Open Governance	**High**: Defined governance process	**Low**: Proprietary software, closed source software	**Medium**: Only open source repository	**Low**: Proprietary closed source software	**Low**: Proprietary closed source software	**Medium**: Only open source repository	**Low**: Proprietary closed source software
Reproducibility of science	**Low**: Without any ease	**Medium**: Data links and scripts shareable without guarantee to be reproducible	**Low**: Without any ease	**Low**: Without any ease	**Low**: Without any ease	**Low**: Without any ease	**Low**: Without any ease
Infrastructure replicability	**High**: Open source code, docker containers, and documentation available	**Low**: Proprietary closed source software	**Medium**: Open source code with basic documentation available	**Low**: Proprietary closed source software	**Low**: Proprietary closed source software	**Undefined**: Dependent on the backend used	**Low**: Proprietary closed source software
Processing scalability	**Medium**: A template application available (Python and Celery)	**High**: Code automatically executed in parallel using a MapReduce approach	**Low**: User runs his own code	**Medium**: HTCondor	**Medium**: A template application available (C++ and MPI)	**Undefined**: Dependent on the backend used	**High**: Closed solution
Storage scalability	**High**: Distributed File System, S3, and HTTP	**High**: Google storage services	**High**: Google storage services	**High**: Distributed File System	**High**: Distributed File System	**Undefined**: Dependent on the backend used	**High**: Closed solution
Data access interoperability	**High**: OGC Services	**Medium**: Tile service	**Low**: Without any ease	**Low**:Without any ease	**Low**:Without any ease	**High**: OGC Services	**High**: OGC Services
Extensibility	**High**: Open source and modular code	**Low**: Proprietary closed source software	**High**: Open source	**Low**: Proprietary closed source software	**Low**: Proprietary closed source software	**Medium**: open source software integrated with proprietary software	**Low**: Proprietary closed source software

tructure used for data storage and processing. In the case of GEE and SH, the user only interacts with their APIs using JavaScript (SH and GEE) or Python languages (GEE) or OGC web services (SH). For OpenEO, physical infrastructure abstraction is one of the main goals of the platform. Thus, it provides a single interface for analysis on different back-ends.

The platforms ODC, JEODPP, pipsCloud, and SEPAL partially abstract the physical infrastructure. Although users have facilities to use ODC, JEODPP, pipsCloud, and SEPAL through remote resources, it is not transparent for users the use of the file system or the use of multiple resources for distributed processing.

ODC, SEPAL, and OpenEO platforms are open source and their code are available in open repositories that allow the scientific community to collaboratively participate in their development. Considering these platforms, ODC is the only one that provide publicly disclose documents that formalize the platform governance process and how to create or incorporate new features into the platform. GEE, JEODPP, pipsCloud, and SH are closed solutions that do not allow collaborative development or participation in governance by members outside the responsible teams.

The only solution reviewed that presents a clear initiative in providing reproducibility of science is the GEE platform. Users can share their scripts and data, allowing other platform users to reproduce their analysis. Nevertheless, the analysis of reproducibility of science is a complex issue when we observe that GEE is a closed solution and that the scripts only represent processing invocations to the platform API. Changes in algorithm implementations, even if they are corrections, can impact the results observed by running the same script on the same dataset at different dates. GEE does not provide a versioning system for algorithms or data in public catalog. For this reason, we consider that GEE provides only an intermediate reproducibility of science capability.

The reproducibility of results in the other platforms is not facilitated through some platform-specific functionalities. In general, users need to manually share scripts and data with other users before the analyzes can be replicated.

ODC, SEPAL, and OpenEO platforms allow users to deploy them on their own infrastructures. ODC, besides providing source code, provides a good documentation about the platform deployment process and a repository with docker images, facilitating the platform infrastructure replication process. Differently from ODC, SEPAL, and OpenEO have little documentation about the deployment of available applications, making this process difficult. The other platforms analyzed do not publicly present means to be deployed in user infrastructures.

Regarding the processing scalability, GEE has the best performance gain with the increase of resources because it uses a MapReduce architecture where data and processing are distributed using technologies developed by Google. Processing scalability can also be observed in the solution adopted by JEODPP in the production of visualization data (processing chains). This same capability is not available in the other data processing modes of the JEODPP, leading us to classify it as medium.

We consider that ODC and pipsCloud partially meet the scalability of processing. For while, these platforms help users to make better use of processing resources by incorporating distributed processing tools or code templates. However, the responsibility to implement such solutions is entirely of the users.

A high level of storage scalability is observed in the GEE, SH, JEODPP, SEPAL, ODC, and pipsCloud platforms. These platforms use or allow the use of distributed file systems to scale data storage. The file systems themselves used by these solutions allow new storage features to be added without a direct impact on how users access data.

Regarding data access interoperability, only ODC, OpenEO and SH platforms provide standardized APIs for accessing stored data. These platforms provide OGC web services, such as WMS, WTMS, and WCS. GEE uses a Tile Map Service (TMS) only to show the results on the web IDE, without explicitly informing the API endpoint.

Only ODC, SEPAL, and OpenEO allow the extension of their functionalities by including new software tools to the solutions already used by them. ODC stands out in this capacity because it provides a good documentation about how to add new features by other users, including the distribution of sample applications for distributed processing.

To summarize and visually illustrate the rating levels assigned to each platform in the analyzed capacities, we produced six graphs. Each graph shows a pair of capacities. When possible, capabilities were grouped by similarities, such as abstraction and scalability issues.

Figure 2.8 presents a graph illustrating the position of each platform in relation to data abstraction and processing abstraction capabilities. In this figure, we observe two groups of platforms, those with low data and processing abstraction (SEPAL, JEODPP, and pipsCloud) and those with high data abstraction and medium processing abstraction (ODC, GEE, and OpenEO).

Figure 2.9 illustrates the distribution of platforms against processing and physical infrastructure abstractions. In this figure, SEPAL, JEODPP, and pipscloud platforms are grouped with low processing abstraction and medium level of physical infrastructure abstraction capability. ODC presents medium classification in both categories presented in this figure.

Figure 2.8 - Data abstraction *vs* Processing abstraction.

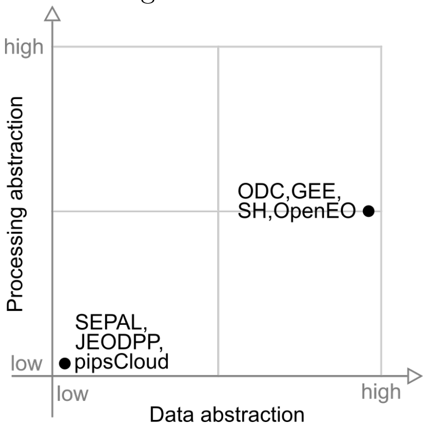


Figure 2.9 - Physical infrastructure abstraction *vs* Processing abstraction.

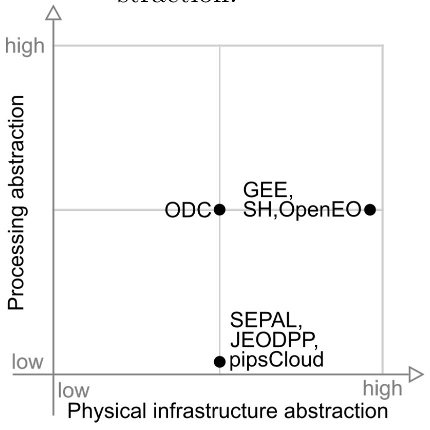


Figure 2.10 presents the position of each platform in relation to reproducibility of science and infrastructure replicability. In this graph we can observe that, in terms of infrastructure replicability, the highlight is ODC, while GEE stands out in the reproducibility of science axis. JEODPP, pipsCloud, and SH have none of these capabilities.

Figure 2.11 presents a graph that shows platforms in relation to storage scalability and processing scalability capabilities. Note that all platforms appear at the top of this figure. SEPAL appears in the top left corner, pipsCloud, JEODPP, and ODC are rated with medium processing scalability and the platforms that appear in the upper right are only GEE and SH.

Figure 2.12 maps the extensibility and data access interoperability capabilities. Among the graphs showed in this Chapter, this graph presents the largest dispersion among the capabilities of the platforms. At the lower left corner, we have JEODPP and pipsCloud, while in the lower right corner only the SH appears. ODC appears at

Figure 2.10 - Infrastructure replicability *vs* Reproducibility of science.

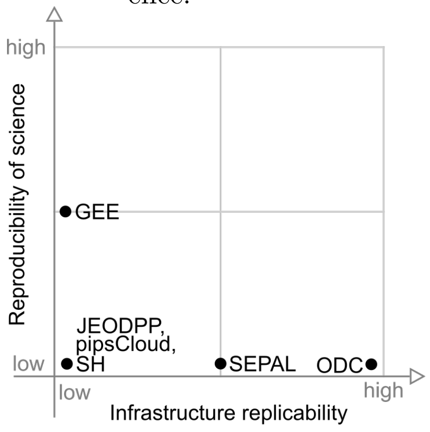
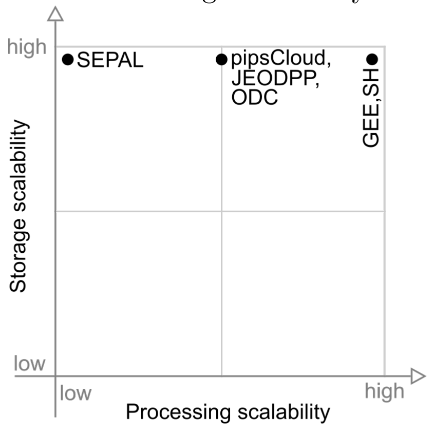


Figure 2.11 - Processing scalability *vs* Storage scalability.



the upper right corner, highlighting its high level in these two capabilities. SEPAL is in the northwest corner, showing good extensibility, while GEE shows a medium data access interoperability.

Figure 2.13 presents a graph relating the extensibility and open governance capabilities. In this figure we can observe four groups. The first is formed by the platforms GEE, JEODPP, pipsCloud, and SH that does not have these capabilities. In the upper part, SEPAL appears at the center of the open governance axis, while ODC appears on the right edge. The latter group appears at the center of both axes and is formed only by OpenEO platform.

Figure 2.12 - Data access interoperability *vs* Extensibility.

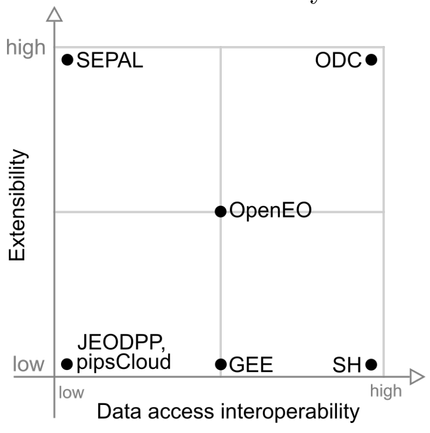
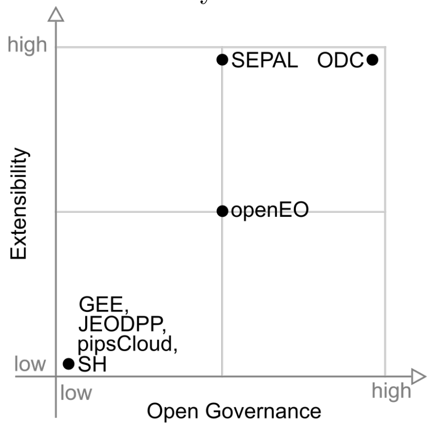


Figure 2.13 - Open Governance *vs* Extensibility.



2.3 Final remarks and discussion

From the point of view of data abstraction and processing, OpenEO is the system that provides the most flexible solution to the scientist. *Process* chaining through *Process Graphs* allows a high degree of abstraction of processing tasks, equivalent to the solution used by GEE. Besides that, OpenEO allows the scientist to use *User Defined Functions*, which allow data processing in different modes (scenes, time series, cubes, and windows). The shortcoming of this processing mode is the lack of support to operate on more than one image.

On the other hand, OpenEO cannot guarantee that back-end systems will provide data, features, and implementations that are compatible with each other. This fact can limit the execution of the same analysis on different back-ends or require researchers to adapt their applications.

From the standpoint of ease of use and development maturity, GEE is the platform that delivers the best solution for users. However, it has limitations because it is a closed platform, especially as to guarantee the reproducibility of the analyzes. Another important issue to consider is the lack of guarantee of platform continuity, as it is not possible to replicate GEE in a particular infrastructure. These drawbacks are also shared by Sentinel Hub.

The technological solution adopted by SEPAL is more focused on infrastructure management and the provision of tools for the analysis of EO data. Big data challenges are not directly addressed by this platform, leaving the user with the burden to make efficient use of available hardware and software resources. The ability to order hardware on demand, as provided by this platform, implies moving data to perform processing. This strategy may be efficient for CPU-bound tasks, but in I/O-bound computations, the approach of moving analysis closer to the data (moving code paradigm) is more appropriate.

Among the analyzed platforms, ODC is the solution that presents the best balance between the analyzed capacities. The drawback of the ODC solution is mainly the lack of support for reproducibility of science, which is not found in the others either. On the other hand, the other capacities evaluated are at least partially met.

Recently, ODC has been used by different projects to manage multidimensional data cubes created from remote sensing images for a specific country, such as the Australian Data Cube (LEWIS et al., 2017), Swiss Data Cube (GIULIANI et al., 2017), and Africa Regional Data Cube (KILLOUGH, 2019). In these projects, Analysis Ready Data (ARD) generated from Earth observation satellite images are modeled as multidimensional data cubes, especially for image time series analysis (NATIVI et al., 2017). Such data cubes have three or more dimensions that include space, time, and spectral derived properties. These data cubes can also be defined as a set of time series associated to spatially aligned pixels ready for analysis (APPEL; PEBESMA, 2019).

This novel worldwide trend in creating multidimensional data cubes from big amounts of remote sensing images brings new challenges for SDI. A novel generation of SDI has to properly store, access, visualize, and analyze these multidimensional data cubes, based on image time series processing. Following this trend, ODC provides functions to index multidimensional data cubes and to handle them, partially meeting these challenges.

Building a platform that fully meets all the capabilities discussed in this Chapter is a big challenge. The greater the degree of abstraction delivered to the scientist, the greater the difficulty in providing flexibility in data processing approaches. The solution, perhaps, is to provide a platform with two forms of processing for the scientist. The most frequently used features could be made available through a high abstraction API, similar to that provided by OpenEO and GEE. For more complex analyzes, the platform could allow its extension through a framework, such as the solution adopted by ODC, so that the scientist has direct access to data and infrastructure processing capabilities. Besides that, the use of distributed data storage is important to minimize data movement during processing. In this approach, processing would occur where data is stored, using the moving code approach. In the scenario presented in this Chapter, we believe that ODC is the solution that presents the best conditions to evolve to a platform with these characteristics.

3 ACCESSING AND PROCESSING BRAZILIAN EARTH OBSERVATION DATA CUBES WITH THE OPEN DATA CUBE PLATFORM[1]

In recent years, the amount of Earth Observation data freely available has grown, motivated by technological advances in acquisition and storage equipment and space agencies' policies that make their data repositories available. The estimated volume of EO data produced in 2019 by Landsat (7 and 8), MODIS (Terra and Aqua units), and the Sentinel missions (1, 2, and 3) exceeded 5 PB (SOILLE et al., 2018). The technological challenges to store, process, and analyze these large data sets impose significant restrictions for EO community scientists to take advantage of all potential of these resources (CAMARA et al., 2016; STROMANN et al., 2020).

A Spatial Data Infrastructure (SDI) provides an environment that allows people and systems to interact with technologies to foster activities for using, managing, and producing geographic data (RAJABIFARD; WILLIAMSON, 2001). In the last years, SDIs have been built using technological components that implement standards proposed by Open Geospatial Consortium (OGC) and Organization for Standardization (ISO) to represent, store, and disseminate spatial data. Even with these standards, most current SDIs are focused on EO data sharing and dissemination in the form of individual files through web portals and HTTP, FTP, and SSH protocols (MÜLLER, 2016).

In the scenario of big EO data, the proper management, processing, and dissemination of this vast volume of data poses a challenge for EO system infrastructures. The needs in this scenario demand more precise and structured research services, automated acquisition, calibration and availability processes, and the possibility of data being processed without having to be moved through the network (WOODCOCK et al., 2016).

To address these challenges, the scientific community recently adopted the Earth Observation Data Cube paradigm, which, through specialized technologies, seeks to change the way researchers deal with these large volumes of EO data (GIULIANI et al., 2019). Even though there is no consensus on the definition of the term EODC, these systems are software infrastructures that manage large time series of EO data using multidimensional array concepts, facilitating the access and the use of Analysis Ready Data (ARD) by users (NATIVI et al., 2017; GIULIANI et al., 2019).

[1]A version of this Chapter as published on the "ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences" (GOMES et al., 2021). The paper is authored by Vitor C. F. Gomes, Felipe M. Carlos, Gilberto R. Queiroz, Karine R. Ferreira, and Rafael Santos.

Many institutions have been developing technologies that use these concepts and can be considered EODC (GIULIANI et al., 2019), such as Open Data Cube (ODC) (OPEN DATA CUBE, 2022b), Google Earth Engine (GEE) (GORELICK et al., 2017), JRC Earth Observation Data and Processing Platform (JEODPP) (SOILLE et al., 2018), Sentinel Hub (SH) (SINERGISE, 2020a), or Brazil Data Cube (BDC) Platform (FERREIRA et al., 2020). These platforms adopt different data abstractions, standards, or technological solutions to provide their functionalities despite similar functionalities (GOMES et al., 2020). ODC and BDC, for example, are both open source solutions that use different data abstractions and methods to process and prepare ARD. BDC Platform uses an R package, called sits (CAMARA et al., 2018) for land use and land cover mapping, while ODC focuses on all data processing and analysis using Python language and xarray package. GEE and Sentinel Hub are commercial solutions that allow users to access and process large EO catalogs using, each one, specialized APIs and different web services. For data abstraction, GEE uses *Image/ImageCollection* and *Feature/FeatureCollection*, while Sentinel Hub uses *Data Source*, *Instances*, and *Layer* concepts.

As a consequence of the difference in the way each platform manages data and provides access and processing to users, there is a lack of interoperability between these solutions, making it a challenge to discover, access, and share processes between EODCs (NATIVI et al., 2017; GIULIANI et al., 2019).

One of the leading platforms in the EODC scenario (GOMES et al., 2020) is the Open Data Cube, which is composed of a set of tools and services for the management, processing, and access to EO data and which has been used by several initiatives and institutions around the world. In July 2019, there were 56 initiatives for ODC, 9 of which are operational, 14 under development, and 33 under review (OPEN DATA CUBE, 2019). The expectation is that there will be 22 operational instances of ODC in 2022 (KILLOUGH, 2018).

In the Brazilian context, the National Institute for Space Research (INPE) leads the development of the Brazil Data Cube Platform for management and analyzing massive EO data. The main objectives of the BDC project are to produce cubes with Analysis Read Data in medium resolution for the Brazilian territory that allows the analysis of time series and the use and development of technologies for processing and storage of those data cubes, including cloud computing and distributed processing (FERREIRA et al., 2020).

This Chapter presents the integration process between the data products of the BDC project and the ODC framework. This integration aims to expand the services and tools that can be used to access, view, and analyze the EODC produced by the BDC project. Besides, the integration intends to allow algorithms previously developed for ODC technology to be more easily adapted for use with the BDC data.

The remainder of this article is organized as follows. In Section 3.1, we present an overview of the Open Data Cube and Brazil Data Cube platforms. Section 3.2 the process used to integrate the systems and the developed tools is presented. Section 3.3 presents the results of the integration of the ODC components with the data and services of the BDC. Finally, in Section 3.4, we make the final considerations regarding the challenges encountered and the next steps that will be developed.

3.1 Earth Observations Data Cubes

Earth Observations Data Cube is commonly used to refer to multidimensional arrays with space, time dimensions, and spectral derived properties created from remote sensing images (APPEL; PEBESMA, 2019). This term is often used to refer to analytical technology solutions that make use of these data structures. The term can also be found to refer to analytical, technological solutions that allow the management, processing, and analysis of these data structures (GIULIANI et al., 2019).

Recently, many institutions have been creating EO data cubes from remote sensing images to specific regions, such as Australian Data Cube (LEWIS et al., 2017), Swiss Data Cube (GIULIANI et al., 2017), Armenian Data Cube (ASMARYAN et al., 2019), Africa Regional Data Cube (KILLOUGH, 2018), Colombian Data Cube (CDCol) (ARIZA-PORRAS et al., 2017), and Brazil Data Cube (FERREIRA et al., 2020). Except for the last one, the other initiatives use the framework Open Data Cube as the core technology to index and handle the EO data. These initiatives use ODC as a starting point and build custom applications to accomplish specific demands. The CDCol initiative, for example, implements tools to handle a bank of algorithms and its live cycle (in development, published, obsolete, and deleted) and user roles to distinguish the responsibilities of users in the platform (ARIZA-PORRAS et al., 2017). On the other hand, the Brazil Data Cube platform uses its own tools to produce and manipulate data cubes, but it has been developing tools to integrate its solutions with the ODC framework, such as the one presented in this work. The following subsections present details about the Open Data Cube framework and the Brazil Data Cube platform.

3.1.1 Open Data Cube

The Open Data Cube is a framework that allows the cataloging and analysis of EO data. It consists of a set of data structures and Python libraries that allow the manipulation, visualization, and analysis of that data. The source code of ODC is available under Apache 2.0 license and is distributed as modules on github[2].

The module datacube-core is responsible for indexing, searching, and retrieving cataloged data. It consists of a Python package and command-line tools that use a PostgreSQL database to store metadata for managed data.

In indexing the data, the description and metadata of a *Product* is initially registered. *Product* is the abstraction used by ODC for data collections that share the same measures (bands) and metadata. Then, information about the *Datasets* that together represent a *Product* are recorded. *Datasets* represent the smallest aggregation of managed data, they usually are scenes of a given *Product* stored in files (OPEN DATA CUBE, 2021).

After indexing the EO data in an ODC instance, it is possible to use an ODC Python API to access and process it. ODC also provides other modules for: data visualization (datacube-explorer); performing temporal statistical analysis (datacube-stats); disseminating data through OGC services (datacube-ows); illustrates the usage of ODC Python interface (datacube-notebooks); etc.

Also, ODC has a catalog[3] with the source code of applications that use the ODC Python API to access data cubes and perform specific analyzes. The Committee on Earth Observation Satellites (CEOS) also provides a repository (CEOS, 2021) with algorithms for processing data accessed in an ODC catalog.

3.1.2 Brazil Data Cube

The Brazil Data Cube project has objectives to produce Analysis Ready Data structured in data cubes in medium resolution for the entire Brazilian territory. For this, the BDC project uses and develops technologies necessary for the processing, analysis, and dissemination of these data products and to produce information on Land Use and Land Cover from the cubes using machine learning methods and image processing techniques (FERREIRA et al., 2020).

[2]<http://github.com/opendatacube>
[3]<https://www.opendatacube.org/dcal>

Figure 3.1 illustrates the services and products of the Brazil Data Cube platform. The first layer, below, represents the generating process of ARD data sets used to create the data cubes. The layer above represents the cataloging of the metadata of the generated data products.

The data and metadata of the data products produced by the BDC are accessed and processed through web services presented in the Services layer. Services using standard protocols are available, such as Tile Map Service (TMS), Web Feature Service (WFS), Web Map Service (WMS), and Web Coverage Service (WCS). In addition to these, there are also special-purpose services developed by the project team, such as Web Time Series Service (WTSS) (VINHAS et al., 2016) and Web Land Trajectory Service (WLTS).

The Brazil Data Cube platform also makes its data products publicly available through a STAC service[4].

The applications, available to end-users, use these web services to access the data cubes produced by the BDC. In this project's structure, ODC behaves like an application that uses BDC services to consume and index metadata and EO data.

All software products developed by BDC are available under MIT license in the project's source code repository[5].

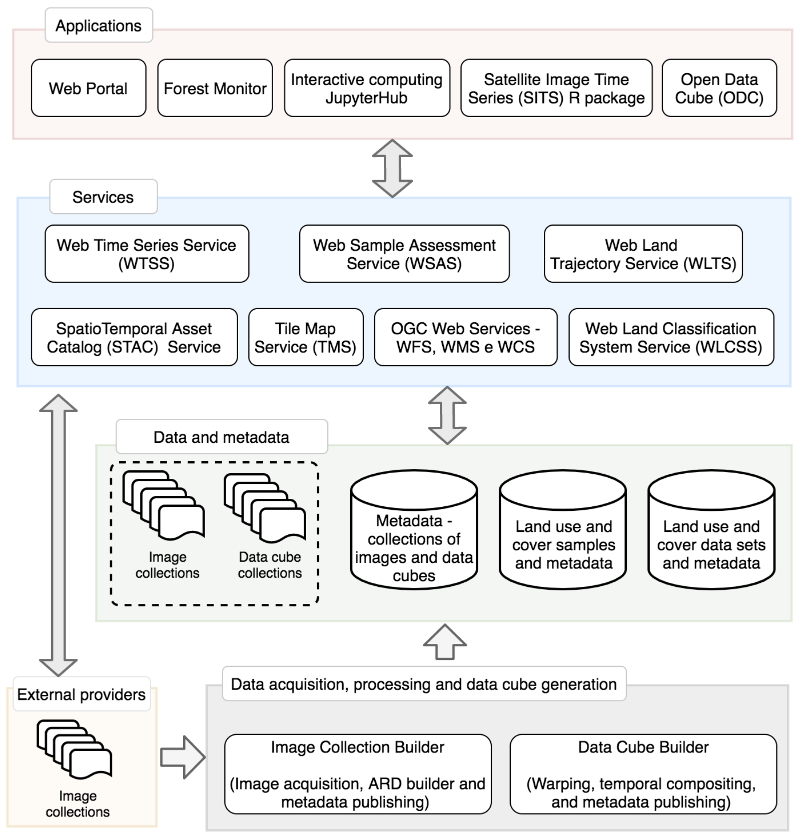
3.2 Methodology

Four steps were necessary to accomplish the integration between BDC's data products in the ODC platform and the availability of a computational environment to use the new integrated functionalities. Figure 3.2 illustrates these four steps and the associated data flow from BDC Platform to the BDC's ODC instance.

The first step was to prepare an ODC instance of the BDC project's computing infrastructure. For this, an instance of the PostgreSQL DBMS was implemented to store the products and datasets catalog metadata (ODC-db). Also, a Docker container (ODC-core) with the datacube-core module of the ODC framework was deployed, which provides the necessary tools to index and manage the metadata catalog. This instance and all the others created in this integration process have direct access to a file system that contains the BDC data repository. Thus, no data replication is required for shared use between the BDC tools and the ODC instance.

[4]<http://brazildatacube.dpi.inpe.br/stac>
[5]<https://github.com/brazil-data-cube>

Figure 3.1 - Brazil Data Cube Project: data and software products.


Applications
Web Portal
Forest Monitor
Interactive computing JupyterHub
Satellite Image Time Series (SITS) R package
Open Data Cube (ODC)
Services
Web Time Series Service (WTSS)
Web Sample Assessment Service (WSAS)
Web Land Trajectory Service (WLTS)
SpatioTemporal Asset Catalog (STAC) Service
Tile Map Service (TMS)
OGC Web Services - WFS, WMS e WCS
Web Land Classification System Service (WLCSS)
Data and metadata
Image collections
Data cube collections
Metadata - collections of images and data cubes
Land use and cover samples and metadata
Land use and cover data sets and metadata
External providers
Image collections
Data acquisition, processing and data cube generation
Image Collection Builder
(Image acquisition, ARD builder and metadata publishing)
Data Cube Builder
(Warping, temporal compositing, and metadata publishing)

SOURCE: Ferreira et al. (2020).

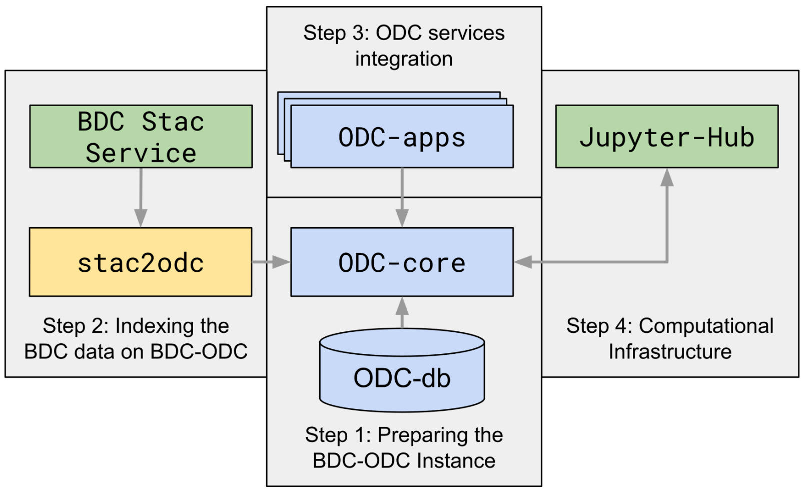
To facilitate readability, we will use the acronym BDC-ODC to refer to the ODC instance running on the BDC project infrastructure.

The other three integration steps are the indexing of BDC data products in the BDC-ODC (Step 2), the adaptation and configuration of ODC framework services in the BDC-ODC (Step 3), and the provision of a multi-user computational infrastructure to access and processing of data indexed in the BDC-ODC (Step 4). The following subsections detail these tasks.

3.2.1 Data indexing

The BDC STAC service was chosen as the source for access the BDC project products' data and metadata. This choice over direct access to the BDC database was motivated by leveraging other future available catalogs' indexing through this spec-

Figure 3.2 - Overview of data flow from BDC data to the BDC-ODC instance.


Step 3: ODC services integration
BDC Stac Service
ODC-apps
Jupyter-Hub
stac2odc
ODC-core
Step 2: Indexing the BDC data on BDC-ODC
ODC-db
Step 4: Computational Infrastructure
Step 1: Preparing the BDC-ODC Instance

ification.

From this choice and motivated by the need to manipulate large volumes of data from the BDC, an application was developed, called `stac2odc`, to automate reading the STAC catalog and indexing the metadata in the BDC-ODC.

Figure 3.2 presents an overview of the data flow used in this integration. It can be seen that the `stac2odc` tool is responsible for collecting the data in the BDC catalog and, after converting it, storing it in the BDC-ODC catalog through the datacube-core module. From that moment on, the data will be available for use in other applications.

The `stac2odc` tool maps the BDC STAC catalog metadata to the format accepted by the ODC. The mapping specification is done through a configuration file. This option allows changes to the BDC or ODC metadata structures to be easily incorporated into the tool.

Listing 3.1 shows an example of the configuration file in JSON format used by `stac2odc` tool. In this example, it is possible to view the mapping definition structure. The JSON keys represent the values' source, and the respective values represent the destination in the ODC metadata file to be generated. This file can also define external scripts (`custom_mapping.py`) for the translation of metadata or constant values.

This configuration file also allows the user to inform the location where the data products consulted at STAC are stored. With this information, `stac2odc` does not need to duplicate the data in the BDC project infrastructure. On the other hand, if it is of interest to the user, it is possible to inform a target folder to download the data being indexed. This feature is handy for researchers who want to populate a particular instance of the ODC with data from the BDC project.

Listing 3.1 - Example of an stac2odc configuration file.

```
{
"product": {
  "fromSTAC": {
  "name": {
    "from": "id",
    "customMapFunction": {
    "functionName": "transform_id",
    "functionFile": "custom_mapping.py"}
  },
  "storage.crs": "bdc:crs",
  "fromConstant": {
    "metadata_type": "eo",
    "measurements.units": "meters"}}}
}
```

3.2.2 ODC services integration

With BDC data products indexed in BDC-ODC, they are now ready for use through the command line tools and the ODC Python API, available on the datacube-core module.

The ODC provides a wide range of tools and services in its ecosystem in a modular way. For this first phase of integrating BDC data products into the ODC, the datacube-ows, data-explorer, datacube-stats, and datacube-ui modules have been chosen. The following paragraphs present the functionalities and considerations made for each module's configuration during this integration.

The datacube-ows module allows the dissemination of data indexed in the ODC catalog through web services in the OGC WMS, WMTS, and WCS standards. Its configuration is done through a command-line tool, which helps to create indexes used by the service in the database. This module also requires creating a configuration file with the description of the products to be published and their associated styles.

Changes to datacube-ows module source code were necessary to use this module with the BDC data products. These changes were made to adapt how a Coordinate Reference System (CRS) is handled in the module internal operations. The BDC project uses a CRS generated specifically for its data products to reduce geometric distortions in the data generated for South American territories (FERREIRA et al., 2020). The CRS used by the BDC project does not have a standard EPSG code, which is required by the original code of the datacube-ows module.

The datacube-explorer module provides a simple web interface for searching data and metadata indexed in the ODC catalog. This application also provides a STAC API for advanced searches. Similar to datacube-ows, this module is also configured using a command-line tool. This tool creates new tables in the database used by the ODC and populates them with the information used by the module. During the configuration of the datacube-explorer, it was also identified the lack of support for the use of CRS that does not have a standard EPSG code. For this reason, it was necessary to modify the source code of the tool in order to make possible the use of custom CRS. This capability was implemented by adding a new command line parameter,`-custom-crs-definition-file`, which allows users to inform the location of an additional configuration file. This file allows linking a CRS, defined through a PROJ String, with an arbitrary EPSG code. These settings are made in JSON format. Listing 3.2 presents the configuration used for the use of the BDC data products in the datacube-explorer module.

Listing 3.2 - BDC custom CRS definition file used in datacube-explorer module.

```
{"epsg:100001": "+proj=aea +lat_0=-12 +lon_0=-54 +lat_1=-2 +lat_2=-22 +x_0=5000000 +y_0=10000000 +ellps=GRS80 +units=m +no_defs"}
```

The need to process a large volume of EO data often requires the researchers to use advanced techniques such as parallel processing and efficient data management in memory. These requirements can represent a barrier for users who want to process and analyze data but do not have enough technical expertise for this complex task.

To address these needs, the ODC `datacube-stats` module provides a simple interface for processing the data indexed in the ODC catalog and automatically managing the computational resources used in this process. In the integration of this module in the BDC-ODC, the configuration of the Dask tool was also done to allow the parallel and distributed analyzes. Dask[6] is a Python library that provides data structures and tools for scheduling tasks across multiple processing nodes.

Finally, the datacube-ui module was configured. Unlike the other modules used, the datacube-ui is available in the CEOS github repository[7]. This module provides a full-stack Python web application for performing analysis on data indexed in an ODC catalog. This application's objective is to provide a high-level interface for users to access the indexed data, perform analyses previously configured in the tool, and easily access the metadata of the analyses performed (CEOS, 2021). This module currently comes with analysis application code such as Cloud coverage detection, Coastal change, Water detection, and spectral indices calculation. It is also possible to include new applications through a template included in the documentation of this module.

Currently, the applications previously available in datacube-ui are not in use in the BDC project due to the incompatibility between the data and metadata required by the applications present in this module and those available in BDC-ODC. The usage of this module is currently more focused on the access and visualization of indexed data and metadata on the BDC project.

The implementation of datacube-stats and datacube-ui modules in the BDC-ODC did not require changes in their source codes.

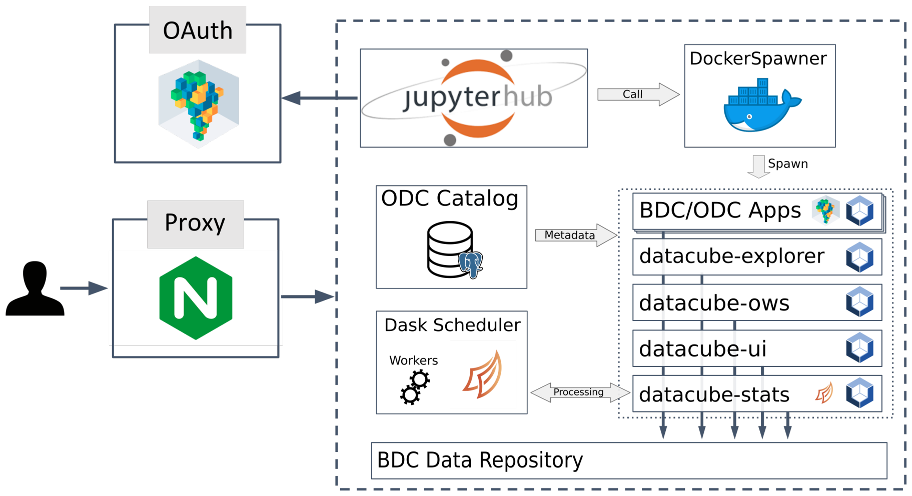
3.2.3 Computational infrastructure

The processing and analysis of the large volume of EO data indexed in the BDC-ODC may require many computational resources. To allow BDC researchers to consume this data, a computational infrastructure was prepared. This infrastructure, illustrated in Figure 3.3, provides a multi-user web interface for processing the data indexed in the BDC-ODC.

The infrastructure presented was done using JupyterHub technology, which provides for each user a ready-to-use isolated environment. The available environments are created by the DockerSpawner module of JupyterHub[8], through Docker images. In these images is stored a set of instructions for creating the environment. Different images can be created and used in the DockerSpawner, which makes the definition

[6]<http://dask.org>
[7]<https://github.com/ceos-seo>
[8]<https://jupyter.org/hub>

Figure 3.3 - Computational infrastructure BDC-ODC.



of environments flexible to users' needs. User authentication is done with the BDC-OAuth service, previously used in other services of the BDC project.

This infrastructure was configured in the data processing servers of the BDC project. For this, Docker images were prepared for the use of datacube-core and datacube-stats tools, being added in the images all the software dependencies of each one of these tools. An instance of Dask Scheduler and Dask Workers was also configured to enable distributed processing in datacube-stats. When an environment is generated by DockerSpawner, it is configured to have direct access to the file system where the BDC data repository is stored. The environment also has access to the database server used by BDC-ODC.

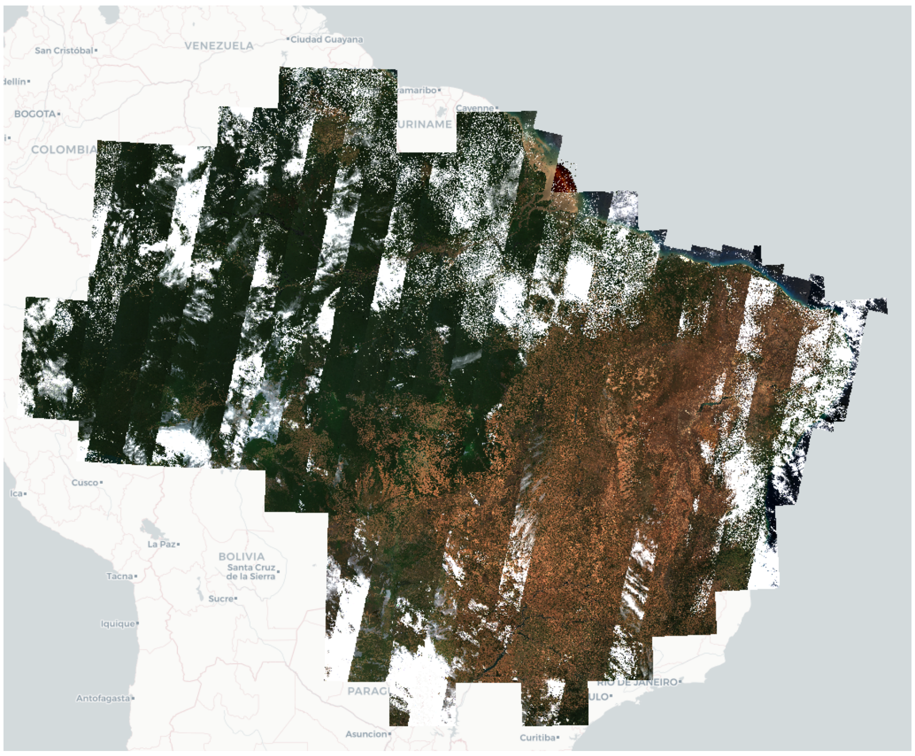
Besides the Docker images for the use of the BDC-ODC, there are also available in the JupyterLab Docker images with other tools used by the BDC project team, such as SITS (CAMARA et al., 2018) and GDALCubes (APPEL; PEBESMA, 2019).

3.3 Results

This section presents the use of the modules and tools made available after the ODC integration in the BDC project platform. Other examples of using the tools presented and the documentation of the changes made can be found in the code repository of BDC project.

The configuration of the datacube-ows module made possible the consumption of the data indexed in the BDC-ODC through the WMS, WMTS, and WCS services. These services bring benefits to several applications by allowing quick access to data, making it possible to perform data processing and visualization. An example of the use of the WMS service can be seen in Figure 3.4. In the example, through a web page, Landsat-8/OLI data from the whole Brazilian territory is presented. The creation of the presented mosaic is done on-the-fly by the datacube-ows module.

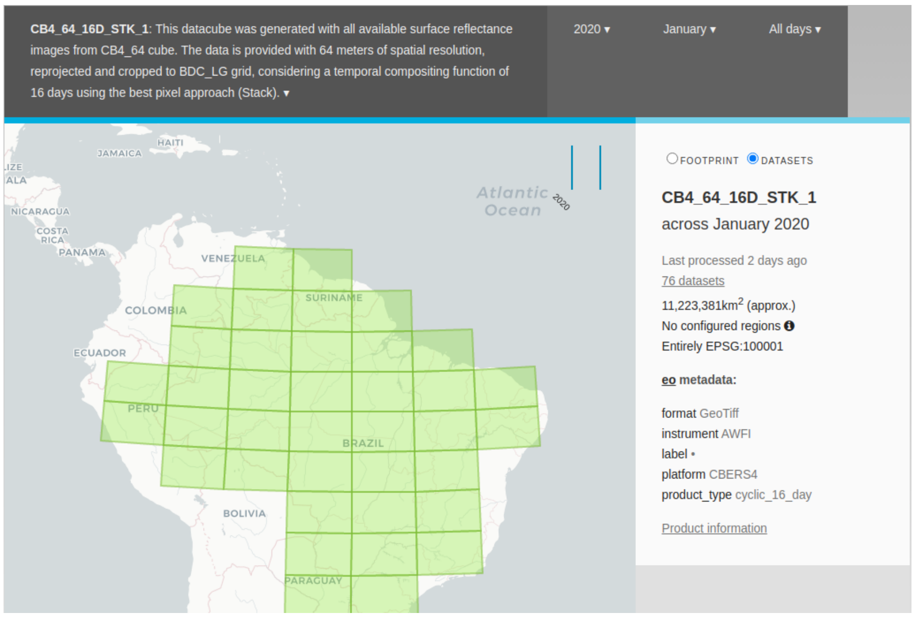
Figure 3.4 - Consumption of data products through WMS via datacube-ows.



The modifications made to the datacube-explorer made it possible to use it to perform space-time searches of the data indexed in the BDC-ODC. All operations can be performed through a web interface, which allows, besides the search, to inspect the metadata of each Dataset identified. Figure 3.5 presents the result for a search made for January 2020 in all Brazilian territory.

For data processing, the datacube-stats tool facilitated the extraction of time statistics from indexed data. With this module, the processing of large spatial extensions

Figure 3.5 - Explorer presenting data from the CBERS-4 collection indexed in BDC-ODC.


CB4_64_16D_STK_1: This datacube was generated with all available surface reflectance images from CB4_64 cube. The data is provided with 64 meters of spatial resolution, reprojected and cropped to BDC_LG grid, considering a temporal compositing function of 16 days using the best pixel approach (Stack). ▾

2020 ▾ January ▾ All days ▾

○ FOOTPRINT ● DATASETS

CB4_64_16D_STK_1

across January 2020

Last processed 2 days ago

76 datasets

11,223,381km^2 (approx.)

No configured regions

Entirely EPSG:100001

eo metadata:

format GeoTiff

instrument AWFI

label •

platform CBERS4

product_type cyclic_16_day

Product information

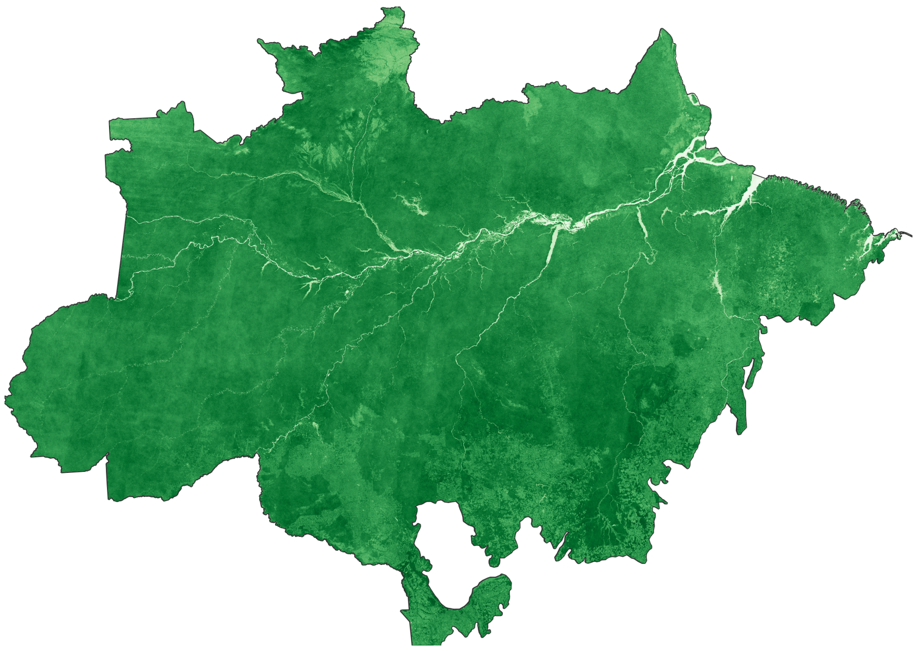
to extract metrics from the data can be done by users without technical knowledge. This tool hides from the user the complexity of the processing distribution. To illustrate the use of datacube-stats, the extraction of the temporal average of NDVI data CBERS-4/WFI was realized in the whole Amazon biome between January 2018 to July 2020. The results are presented in Figure 3.6.

The extraction of time metrics from the data, presented in Figure 3.6, was performed in the prepared computational infrastructure based on JupyterHub. The possibility of adding ready-to-use environments, easily customized with the users' needs, made it simple to apply the indexed data in different contexts. To show another customization of this infrastructure and the use of existing applications for the ODC, Figure 3.7 presents the result of the execution of a code developed by CEOS researchers to cluster pixels of a Sentinel-2/MSI scene (CEOS, 2021). The scene chosen for this analysis is located in the state of Roraima, in Brazil.

3.3.1 Code and data availability

All the tools necessary to reproduce the activities presented in this work are available in the repository <https://github.com/brazil-data-cube/bdc-odc>.

Figure 3.6 - Temporal NDVI Mean from Landsat-8/OLI collection in 01/2018 to 07/2020.



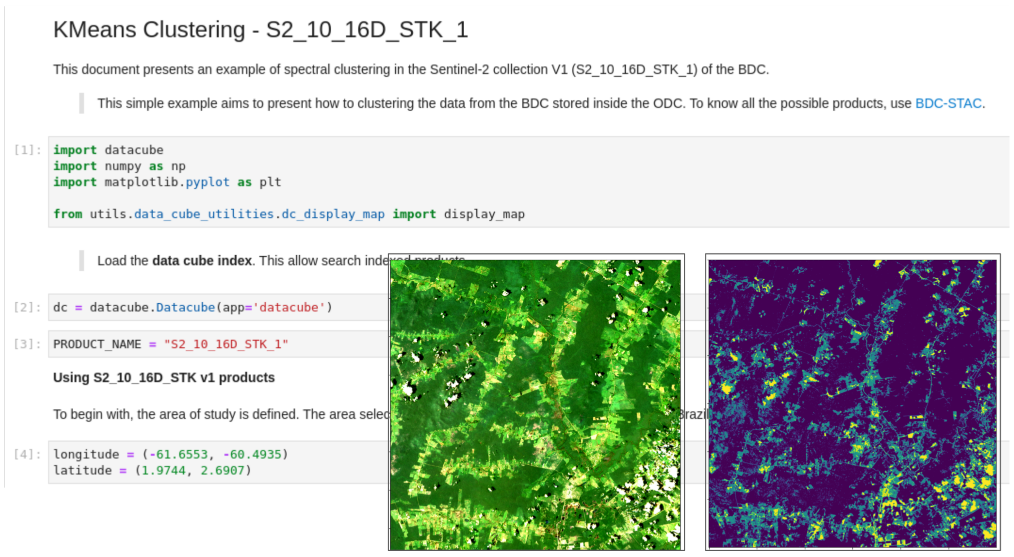
In this repository, the following are available: i) the source code of the `stac2odc` tool and usage documentation; ii) the modified ODC modules; iii) documentation of changes made to the ODC modules; iv) the configuration files used in the deployment of ODC modules; and v) Dockerfiles to generate the images used in this integration. We believe that the content of this repository allows an interested researcher to prepare an ODC instance in his local infrastructure with the data available in the BDC's STAC catalog, including the modules presented in our integration.

The added usage documentation was created following (KILLOUGH, 2018) recommendations. Thus, details have been passed on so that others can understand and consume the lessons we have learned during the integration process.

3.4 Discussion and final remarks

In this work, we presented the integration process between the Open Data Cube framework with the Brazil Data Cube project's data products. The results obtained indicate that the integration approach, done through the conversion of metadata formats and linking with the existing data repository, can be used as an initial form of interoperability between the technologies. The tool created for such a process is

Figure 3.7 - Computational infrastructure used to clustering Sentinel-2/MSI pixels in Roraima state, Brazil.



KMeans Clustering - S2_10_16D_STK_1

This document presents an example of spectral clustering in the Sentinel-2 collection V1 (S2_10_16D_STK_1) of the BDC.

> This simple example aims to present how to clustering the data from the BDC stored inside the ODC. To know all the possible products, use BDC-STAC.

```
[1]: import datacube
     import numpy as np
     import matplotlib.pyplot as plt

     from utils.data_cube_utilities.dc_display_map import display_map
```

> Load the **data cube index**. This allow search inde

```
[2]: dc = datacube.Datacube(app='datacube')
```

```
[3]: PRODUCT_NAME = "S2_10_16D_STK_1"
```

Using S2_10_16D_STK v1 products

To begin with, the area of study is defined. The area selec Brazil

```
[4]: longitude = (-61.6553, -60.4935)
     latitude = (1.9744, 2.6907)
```

independent of the source of the data. This tool can be used to consume other data services that implement the STAC specification.

The addition of the ODC ecosystem modules to the BDC-ODC complemented the BDC's technology base to disseminate and process the large volumes of data generated by the project. With the use of datacube-ows, the BDC data cubes became available through standardized and interoperable interfaces, such as OGC WMS, WMTS, and WCS. These services allow researchers use well-known geographic information systems (GIS) to consume these data.

In this study, an example of using the datacube-stats for the processing of NDVI time averages for the entire Amazon biome was presented. This tool, the functionalities of the datacube-core module and the computational infrastructure defined in this work, can be used as a reference setup for the application of *time-first, space-later* approaches that analyses the temporal variation of EO datasets.

Even with the benefits presented and the ODC tools' maturity, the integration required adaptations to some modules' source code for their adequacy and use with the BDC data. Such modifications were made to the datacube-ows and datacube-explorer modules, which out-of-the-box expected data that had a coordinate reference system (CRS) with a standard EPSG code. The modifications made improve the ODC ecosystem tools, making their use possible in more general scopes of data cubes. However, further testing is needed to verify that any defined coordinate reference system is compatible.

Although the process of this work still represents the beginning of the integration process between ODC and BDC technologies, we believe that there are two main contributions presented: the provision of data from the BDC project to the EO community already familiar with ODC tools; and the possibility for users of the BDC platform to take advantage of the large catalog of algorithms available for the ODC framework.

For the next steps, we hope that more tests will be done on the modifications made to the modules to be sent to the official ODC repositories. It can help the ODC community reach out to more projects and initiatives that seek to use the tools available. It is also essential to consider that during integration, modules such as datacube-ui, due to attribute incompatibilities, could not be readily used to apply the algorithms to the indexed data. It is expected that the adaptation or even the development of new algorithms will be performed for this tool.

Although there is still a long way to go for optimal interoperability between EODCs, this work of integrating ODC tools with the BDC platform paves the way to increase integration between these technologies. Currently, few efforts are known in the domain of EODCs integration. These initiatives are important to prevent these solutions from becoming silos of information (GIULIANI et al., 2019).

4 BRAZIL DATA CUBE WORKFLOW ENGINE: A TOOL FOR BIG EARTH OBSERVATION PROCESSING[1]

4.1 Introduction

Earth observation (EO) data are currently instrumental in comprehending the processes that occur on our planet, leading to significant advancements in monitoring environmental changes, risk detection, urban occupation, and food security (BROWN, 2016). By extracting information from EO data, researchers and policymakers can formulate and implement effective policies for protecting the environment and managing natural resources sustainably.

Satellite observations and geospatial data are being obtained and shared at an unprecedented rate with petabyte production on a daily basis (PAGANINI et al., 2022). Storing, processing, and analyzing these vast datasets pose significant technological challenges that limit the ability of EO scientists to capitalize on their potential (CAMARA et al., 2016; STROMANN et al., 2020). These datasets often exceed the storage, processing, and memory capacities of personal computers, leading users to utilize only a fraction of the available data for scientific research and operational applications (MÜLLER et al., 2010; CAMARA et al., 2016; STROMANN et al., 2020). Thus, novel technological solutions are required to adequately store, process, disseminate, and analyze these large EO datasets.

The Spatial Data Infrastructure (SDI) provides an environment that fosters the use, management, and production of geographic data by allowing people and systems to interact with technology (RAJABIFARD; WILLIAMSON, 2001). In recent years, SDIs have implemented technological components that adopt the standards proposed by the Open Geospatial Consortium (OGC) and International Organization for Standardization (ISO) to store, represent, and disseminate spatial data. However, most current SDIs primarily focus on sharing and disseminating EO data as individual files through web portals and various protocols, such as HTTP, FTP, and SSH (MÜLLER, 2016).

In the context of big EO data, managing, processing, and disseminating this enormous amount of data poses a significant challenge for EO system infrastructure. This scenario demands more structured and precise research services, automated acquisition, calibration, and availability processes, as well as the ability to process

[1]This chapter is a unpublished manuscript entitled "Brazil Data Cube Workflow Engine: a tool for big Earth Observation data processing".

data without the need to move them across the network (WOODCOCK et al., 2016).

In response to these challenges, the EO community has developed new technologies in the form of platforms for big EO data. These platforms are computational solutions offering a range of functionalities for managing, storing, and accessing big EO data. These platforms allow for server-side processing, eliminating the need to download massive amounts of EO datasets. In addition, they provide a certain level of data and processing abstractions that are useful to EO community users and researchers (GOMES et al., 2020). These platforms integrate different types of technologies, Application Programming Interfaces (APIs), and web services, resulting in a more comprehensive solution for managing and analyzing big EO data.

Many institutions have adopted platforms for big EO data, such as Open Data Cube (ODC) (OPEN DATA CUBE, 2022b), Google Earth Engine (GEE) (GORELICK et al., 2017), JRC Earth Observation Data and Processing Platform (JEODPP) (SOILLE et al., 2018), Sentinel Hub (SH) (SINERGISE, 2020a), pipsCloud (WANG et al., 2018), and openEO platform (EUROPEAN SPACE AGENCY, 2022). These platforms adopt different data abstractions, standards, or technological solutions despite their similar functionalities.

In a previous work (GOMES et al., 2020), we performed a review and comparative analysis of these platforms in relation to ten capabilities, including governance, infrastructure, data and processing abstractions, and extensibility. We discussed that the greater the degree of abstraction delivered to the scientist, the greater the difficulty in providing flexibility in data-processing approaches. EO platforms need layers of abstractions that enable both data scientists and data production staff to express computations that exploit available computational resources. One possible alternative would be to provide scientists with a platform that provides two ways to perform server-side data processing. In the first form, an API with a high level of abstraction is made available for scientists to describe their analyses in a manner equivalent to that provided by GEE or OpenEO. The second method allows new algorithms to be added to the platform. These algorithms would directly access the data and take advantage of the distributed processing capabilities provided by the platform.

In the Brazilian context, the management and analysis of massive EO data are made from the Brazil Data Cube project, which has led to the development of the Brazil Data Cube Platform (BRAZIL DATA CUBE PROJECT, 2022). This project is responsible for the production of ARD cubes for the entire Brazilian territory and the development and availability of the BDC platform, with services and tools for accessing, analyzing, and processing the produced data cubes (FERREIRA et al., 2020)

Currently, most of the data generated by the BDC platform are produced using two applications developed by the project team. These applications, called *BDC Collection Builder* and *BDC Cube Builder*, are configured by platform maintainers to discover and retrieve scenes from EO data providers, index them in collections, and produce ARD cubes. These applications are configurable through the definition of processing workflows to enable the production of different types of products. This process is performed by defining a structure in JSON format that specifies the selection, parameterization, and chaining of a set of operations previously implemented in these tools (MARUJO et al., 2022). The BDC platform maintainers have two versions of these tools. One runs on AWS using Lambda services, and another runs on BDC on-premise servers (FERREIRA et al., 2022).

The users of the BDC platform can perform their analyses in two ways. The first, available to the public, consists of downloading the data and software products from the BDC and performing processing and analysis on a personal computer or in any other environment external to the BDC's infrastructure. The second option, currently available to researchers associated with the project, involves using a Jupyterhub environment available on the BDC platform. In this interactive environment, scientists can develop and run scripts to process and analyze EO data using the on-premise servers of the BDC project. One of the tools used by specialists in this environment is SITS (SIMOES et al., 2021). It is an open-source R package that is used for satellite image time-series analysis and LULC map generation. This package uses parallelization techniques to speed up the processing of datasets. However, there is no native support for large-scale processing of clusters of computers, as available in the *BDC Collection Builder* and *BDC Data Builder* tools.

The integration of the ODC framework into the portfolio of services and tools offered by the BDC project was one of its initiatives to broaden the available processing tools accessible to users. For this integration (GOMES et al., 2021), a tool for importing data was developed, and ODC modules were adapted to support the data produced by the BDC project. As a result, this integration is made available to BDC users: i) the ODC API in the BDC JupyterHub environment; ii) services for viewing metadata and data (datacube-explorer and datacube-ows); and iii) the datacube-stats tool, which allows parallel processing of scenes recorded in an ODC catalog.

Similar to the *BDC Cube builder*, the datacube-stats is a command line tool that provides a set of previously defined statistical processing, but allows new processing functions to be added from the extension of the `Statistic` class and the implementation of two new methods: – `Measurements`, which provides a list of measurements that the class will produce, and – `compute`, which takes a `xarray.Dataset` and returns a `xarray.Dataset` with the computed measurements (OPEN DATA CUBE, 2022a). Scene processing is described using a YAML file.

A common characteristic observed in these solutions is the approach used to describe the processing tasks. The explicit or implicit use of Directed Acyclic Graphs (DAGs) to represent workflows has been observed in GEE, OpenEO, BDC Collection and Cube Builders, and ODC Stats. While BDC and ODC uses text files to describe processing flows, GEE and OpenEO provide a higher-level interface, allowing the user to write code in a programming language (Javascript and Python for GEE and Javascript, Python, and R for OpenEO). In both cases, these scripts are converted into data structures that represent DAGs and are sent to run on the backend. Using this technique of sending code close to the data, known as the Moving Code approach (MÜLLER, 2016), EO platforms are evolving to integrate data ready for analysis and technologies for extracting information on the server side.

Based on the knowledge acquired in our previous works (GOMES et al., 2020; GOMES et al., 2021) and on BDC demands to enable users and developers of the BDC project to describe sequences of processes to be efficiently executed in the project server-side infrastructure, we proposed a system architecture to be integrated in the BDC platform. This system uses DAGs as a core concept and integrates the OpenEO API to allow the submission and control of processes by the users. To take advantage of all the computational power available in the infrastructure of the BDC project, this system uses technologies for orchestrating processes distributed in clusters of computers.

The remainder of this section is organized as follows. In Section 4.2, we present the main concepts adopted, the proposed architecture, and the details of the implementation of a prototype, called Brazil Data Cube Workflow Engine (BDC-WE). Section 4.3 presents two study cases. In the first one, legacy processing flows from INPE's Mapaquali project were converted to DAGs to be processed in the BDC-WE. In the second case study, the processing flow of a land use and land cover classification application written in R language using SITS was converted into DAGs and executed using the BDC-WE. Finally, in Section 4.4, we present the final considerations and the subsequent steps that will be carried out.

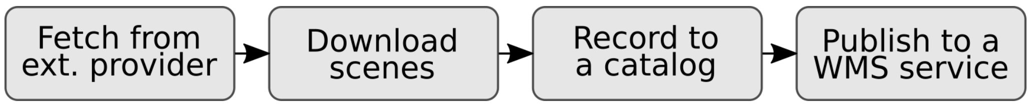
4.2 BDC-WE: A tool for big EO processing

The architecture presented in this section uses workflow as a central concept for the description of processing described by the chaining of tasks. Direct Acyclic Graphs are used to represent the workflows. In this approach, each vertex of a graph represents a specific operation and the edges indicate the data dependency between each operation.

In this study, the nomenclature used by OpenEO (OPENEO, 2022) is used as a reference, where the vertices (tasks) are called *Process(es)* and the chains of *Process* (DAG) are called *Process graph(s)*(*PGs*).

Figure 4.1 illustrates a simple example of *PG* with four *Processes*. This workflow, which illustrates data collection from an external provider, includes *Processes* for: i) scene discovery from an external provider; ii) downloading the scenes to a local repository; iii) registration of new scenes in a metadata catalog; and iv) the publication of new scenes in a Web Map Service (WMS).

Figure 4.1 - Process Graph example.

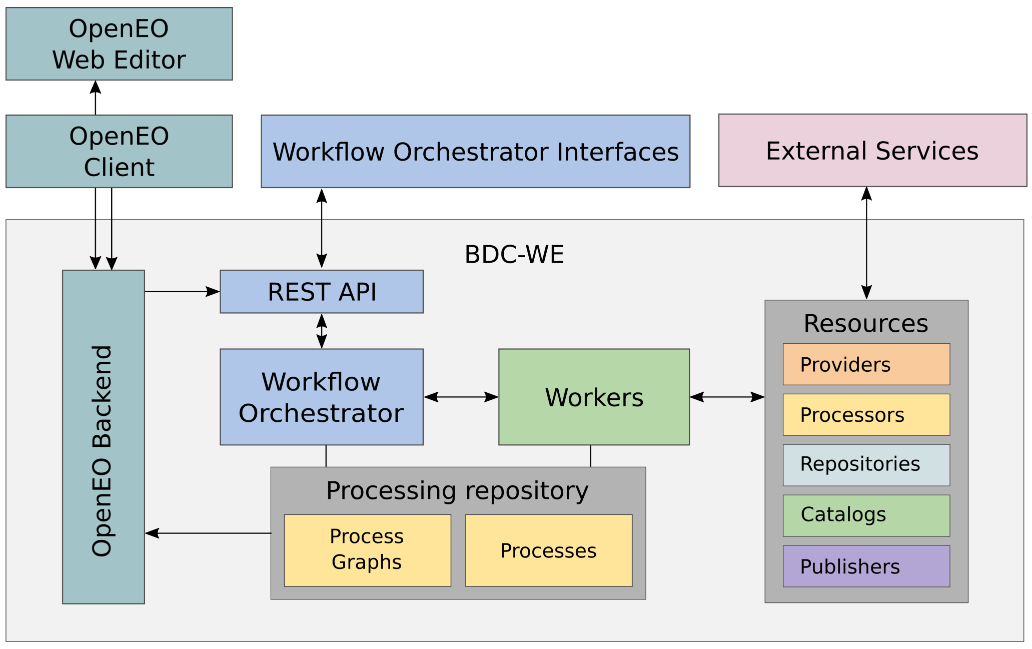


In the context of the BDC-WE architecture, *Process* represents a meta-task, which represents an operation class that does not have a functional core that will actually perform the expected operation. BDC-WE uses an abstraction called *Resources* to configure the *Process* with the operations to be performed.

Figure 4.2 shows a diagram of the BDC-WE architecture. Some elements of this diagram represent software artifacts (*Resources*, *Processing repository*, and *OpenEO Client*), tools for workflow orchestration and task execution (*Workflow Orchestrator* and *Workers*), (web)services (*OpenEO Backend*, *Rest API*, and *External Services*), and Graphical User Interfaces (GUIs) (*OpenEO Web Editor* and *Workflow Orchestrator Interfaces*).

To make the rest of this text easier to read, we use the following terms to refer to the actors that interact with a BDC-WE instance: i) *Developers*: people responsible for deploying or maintaining a BDC-WE instance. Have technical knowledge for configuring BDC-WE and the other tools/services used; ii) *Experts*: people who master the topic of data products that are produced on the BDC-WE platform. They are responsible for the creation/parameterization of the algorithms that generate the data products; and iii) *Users*: people who make use of the services, APIs, or GUIs available in a BDC-WE instance. These actors do not need to have technical mastery of the inner workings of BDC-WE.

Figure 4.2 - BDC-WE diagram.



Resources are software artifacts (classes or functions) that abstract elements managed by the platform and that provide the implementations of the processing that will be performed. They must be accessible to *Workers* so that they can be instantiated and passed as a dependency on *Processes*.

For example, to record scenes in a local catalog, a new *Resource* can be implemented by extending an interface called `Catalog`. The use of *Resources* prevents the platform from having to know the algorithms that will be used and expands the opportunities for use in different applications, only observing that the specificities of each solution are integrated.

Currently, BDC-WE manages the following types of *Resources*.

- ***Provider***: represents an external provider. It has functions for searching scene metadata in an external catalog;
- ***Catalog***: represents a catalog of metadata that can be managed by BDC-WE. It provides functions for searching, adding, and removing scenes in a catalog;
- ***Processor***: represents a processing function that can be applied to a scene or a set of scenes;
- ***Publisher***: represents an external service to the platform. Provides functions for publishing (and unpublishing) a scene or a set of scenes;
- ***Repository***: represents a file system manager where data is retrieved or written. Provides functions to manage the structure of directories where the data will be stored; and
- ***Features***: represents a collection of vector data that can be used as input parameters to *Processors*. Provides a catalog of vector data.

The *Processing Repository* represents the repository with functions and classes that implement the available *Processes* and descriptions of *Process Graphs* available in the platform. `Developers` can add new *Processes* to the list of built-in *Processes* available in BDC-WE.

Workers are responsible for executing *Processes*. The use of multiple *Workers* can increase the scalability of processing large volumes of data.

The *Workflow Orchestrator* (*WO*) is responsible for managing the execution of *Process Graphs* performed by *Workers*. It is responsible for loading the available *Process Graphs* and *Process*, checking whether the input and output dependencies between the *Process* are compatible, receiving execution requests from *Workflow Orchestrator Interfaces* or *OpenEO Backend* through *BDC-WE REST API*, and managing the execution of the workflow on *Workers*.

BDC-WE REST API is a module responsible for managing access and identifying users who will submit and monitor processing. For example, it is used to limit access to restricted data or the number of running processes by a *User*.

The *openEO Backend* is responsible for providing a standardized API so that external clients to the BDC-WE can interact with the platform using available libraries or graphical interfaces, such as the openEO client (JavaScript, Python, and R) and the openEO Web Editor. The *openEO Backend* is responsible for making *Processes* available in the BDC-WE instance so that users can describe their *Process Graphs* using an openEO client. This backend also allows *Users* to invoke the *Process Graphs* to run and download the produced data.

Workflow Orchestrator Interfaces (*WOI*) represents interfaces that allow interaction with *WO*, allowing the configuration and execution of *Process Graphs*. They are mainly intended for `Developers`, as they require technical mastery of how BDC-WE works and provides more details about *Process Graphs* and processing executions. *WOI* can be used for scheduling recurring processing.

The **External Services** represents the services used by the `Resources` in a BDC-WE instance, for example, an STAC Provider, an OGC WMS, etc.

In addition to *Resources*, which represent the resources that perform processing, BDC-WE provides a set of classes that represent processable elements. These data models have the necessary attributes to be managed by *WO* and can be extended by developers to include other attributes or methods.

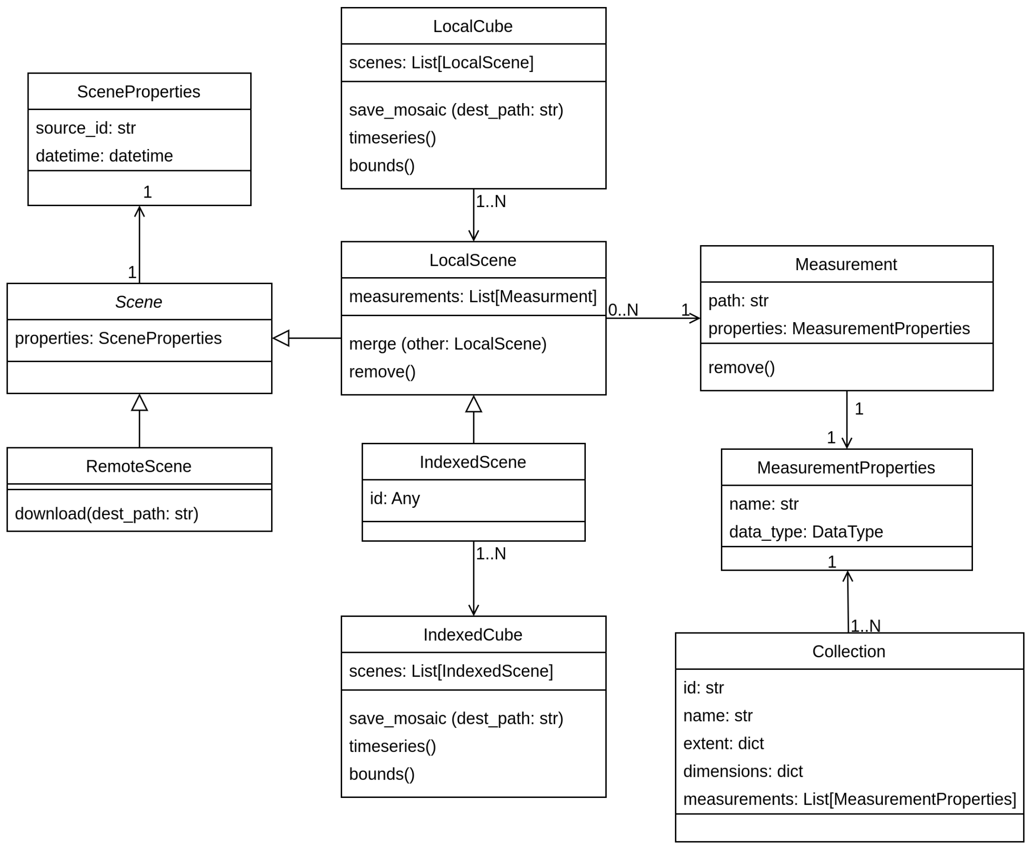
Figure 4.3 presents a diagram of the classes supported by BDC-WE. The core element is `Scene`, which is extended to `LocalScene` and `RemoteScene`. A `LocalScene` represents a `Scene` that has a list of `Measurements` and methods to combine with other `LocalScenes` (`merge`) or to remove it (`remove`). A `Measurement` has a `path` to a file and a `MeasurementProperty`. A `MeasurementProperty` has at least two attributes: `name` and `data type`. A `RemoteScene` uses a method `download` that implements a way to download files from a `Scene` to the repository. An `IndexedScene` extends `LocalScene` by including a unique identifier for the `Catalog` in use. A `Collection` represents a set of `Scenes` that share the same `MeasurementProperties` types. A `Cube` is defined as a set of `Scenes`.

4.2.1 Implementation

To implement the architecture described in the Section 4.2, a set of technologies were chosen as a way to accelerate the framework development process and reuse open-source solutions that met the needs of the BDC-WE.

The Python language was used as a reference, as it is used for BDC Plataform for

Figure 4.3 - BDC-WE data abstraction model.



collection and cube Builders and is also adopted by other platforms, such as Open Data Cube, openEO, and Google Earth Engine (GOMES et al., 2020).

The core element of the framework is the *Workflow Orchestrator*. In the ecosystem of processing data through workflows, are available a variety of open-source tools, such as Apache Airflow[2], Argo[3], Temporal[4], and Dagster[5].

Apache Airflow is a platform that allows the programmatic creation of workflows in Python and the scheduling and monitoring of executions. *DAGs* are defined through the instantiation of `Operators` available on the platform. A *DAG* is created from the dependency configuration between `Operators`.

[2]<https://airflow.apache.org/>
[3]<https://argoproj.github.io/>
[4]<https://temporal.io/>
[5]<https://dagster.io/>

Argo, however, has a higher granularity for *tasks*. This engine manages workflows in a Kubernetes environment, where each *task* is represented by the execution of a container and a workflow is defined through a YAML file.

Temporal is a platform for orchestrating workflows written in Go, Java, PHP, Python, or TypeScript codes. In Python, a *DAG* is represented by a class decorated with a decorator, `@workflow.defn`, and each *task* is represented by a method decorated with `@activity.defn`. Temporal is a platform that is still under development and does not have full support for some languages. The tool documentation is also under development (TEMPORAL TECHNOLOGIES, 2022).

Dagster is a platform for orchestrating workflows in Python. The elements that constitute the processing workflow are defined using the decorators available in the `dagster` package. A workflow *task* is a function decorated with the `@op` decorator, and a *DAG* is a function decorated with `@graph` called a task function. By identifying the *task* calling sequence, Dagster creates a *DAG* structure with data dependencies between the *tasks*. This structure is used during the orchestration of workflow execution. In Dagster, a workflow can run locally in serial or parallel modes using DASK, Celery, Docker, or Kubernetes. Triggering the execution of a DAG can be performed through a WEB interface, GraphQL API, Python code, or by configuring Schedulers or Sensors. Dagster also has the ability to export a DAG to run on Apache Airflow. Dagster provides a paid service to run DAGs in a private cloud environment (ELEMENTL, 2022).

Dagster (ELEMENTL, 2022) was chosen for use as a *WO* on the BDC-WE platform. This choice was made because it allows the dynamic generation of *Process Graphs* and provides a GraphQL API for the interaction between external applications and the orchestrator. This API is necessary for the interaction between the *WO*, *WOI*, *openEO Backend*, and *BDC-WE Rest API*.

In addition, Dagster has a web graphical interface that allows an easy configuration and monitoring of the processes and a variety of execution modes, allowing the execution of all the processing locally, in a single thread or in multiple threads, or the distribution of the processing, using Celery, Dask, and/or Kubernetes technologies. These features facilitate the debugging process and the transition between development and operational environments.

In Dagster, the central processing unit is called `Ops` and represented through functions with the `@op` decorator. In this way, all *Process* developed for BDC-WE use this decorator with the respective metadata to ensure compatibility verification of function input and output parameters. *Process Graphs* in Dagster are called `Graphs` and are defined using functions decorated with `@graph` and making calls to `Ops` functions .

The dynamic generation and configuration of these elements can be achieved through the classes available in the `dagster` package. Using this functionality, BDC-WE can generate these elements from a JGF (Json Graph Format) file that describes a *Process Graph*. The objective of this feature is to facilitate the configuration and maintenance of the workflows managed by the BDC-WE. These files follow a high-level format, without requiring technical knowledge about how Dagster works. In the Section 4.3, an example of the use of this functionality is presented.

BDC-WE-API was implemented through a REST service that intermediates requests from *WO* and *openEO Backend* to *WO*. REST requests are converted to GraphQL requests, which interact with the Dagster service. In the current phase of BDC-WE development, only access control via username and password is performed by *BDC-WE REST API*. Controlling the number of running processes or limiting the *Process Graphs* available for each *User* is not implemented yet.

For the development of the OpenEO Backend, a template available in the repository[6] of the developers of the openEO standard was used as a starting point. It implements the general REST request handling of the openEO API and dispatches the work to a pluggable openEO backend driver. Thus, a driver was developed to translate requests from openEO API requests into requests for *WO*. This driver uses the same *Resources* used by *Processors* running on *Workers*. `Resource Catalog`, for example, is used by developed driver to find available collections or find scenes to be delivered to users.

The *openEO backend* interacts with *WO* through the GraphQL API available in Dagster. For this, a Python client was developed (`dagster_graphql_client`[7]). This client allows starting runs, tracking run status, retrieving results, and reloading *Process Graphs* available for running. This last feature, together with the possibility of generating new *Process Graphs* at runtime, provides great flexibility when creating new workflows. The implemented GraphQL client also has the possibility of controlling and managing executions through command line.

[6] <https://github.com/Open-EO/openeo-python-driver>
[7] <https://github.com/vconrado/dagster_graphql_client>

The BDC-WE is still a prototype and the *openEO Backend* developed still does not support all functions of the openEO standard. New functions are being incorporated according to the demand of Developers who use BDC-WE in INPE's internal projects.

A minimal working set of *Resources* is available in BDC-WE prototype. These built-in *Resources* can be used in operational applications or serve as a reference for other implementations. The built-in *Resource* `stac_provider` available in BDC-WE, for example, extends the `RemoteScene` class to create the `StacRemoteScene` class, which implements the downloading of scenes listed in a STAC service. *Developers* can develop new *Resources* from the inheritance of base classes made available in BDC-WE.

Table 4.1 presents a list of *Resources* currently available in BDC-WE.

Table 4.1 - Resources available in the BDC-WE.

Name	Type	Description
stac_provider	Provider	Performs queries in a STAC service.
odc_catalog	Catalog	Performs insert, remove, and search metadata operations in an Open Data Cube catalog.
crop_scene_proc	Processor	Crops a Scene using GDAL.
reduce_time_proc	Processor	Performs time reduction operations (max, min, mean, median) on a cube.
download_proc	Processor	Download a remote scene via HTTP.
geoserver_pub	Publisher	Publishes a scene to an ImageMosaic Store on a Geoserver server.
ssh_pub	Publisher	Run a previously configured ssh command.
discord_pub	Publisher	Sends a message to a Discord channel.
slack_pub	Publisher	Send a message to a Slack channel.
file_system_repo	Repository	Performs the creation of folders in the file system using the metadata of the scenes.
protected_file_system_repo	Repository	Performs the creation of folders in the file system, blocking the other folders for read-only.
gdal_features	Features	Loads and serves vector data in formats supported by GDAL.

In addition to *Resources* and data models, BDC-WE prototype also provides a set of previously implemented *Processes* that can be used by *Developers* to build *Process*

Graphs. If necessary, *Developers* can implement new *Processes* and make them available to the platform through a configuration file. With the currently available *Processes*, we believe that a large number of EO data processing applications demanded by INPE projects can be modeled because these *Processes* represent meta-tasks and that the code to be executed basically depends on the *Resources* used.

The *Processes* available in the framework are grouped into 4 types:

- ***Discovery***: *Processes* that allow discovery of the resources to be processed. The discovery can be performed in external services (*Provider Resource*) or in the catalog managed by BDC-WE (*Catalog Resource*). Discovery functions in *Providers* produce `RemoteScenes` while discovery functions in *Catalogs* produce `IndexedScenes`;
- ***Processing***: *Processes* that call a processing function (*Processor Resource*) which must receive a `LocalScene` or a `LocalCube` and will produce, respectively, a new `LocalScene` or a new `LocalCube`;
- ***Indexing***: *Process* that registers a `LocalScene` or *LocalCube* in the catalog managed by the BDC-WE (*Catalog Resource*). The indexing process takes a `LocalScene` or `LocalCube` and produces, respectively, an `IndexedScene` or a `IndexedCube`;
- ***Publishing***: *Process* which notifies an external service (*Publisher Resource*) about the creation or removal of a scene or set of `Scenes`. A *Publisher* receives a set of *IndexedScene* or an *IndexedCube* and must return an object of the same type received.

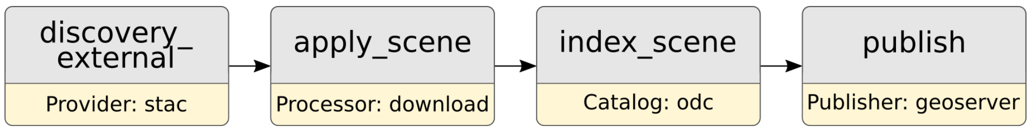
Table 4.2 presents the *Processes* currently available in the BDC-WE prototype. In this Table, the first column presents the name of the *Process*, the second the type, and the third the *Resources* used. The fourth and fifth columns present the types of object expected as the input and output by *Resource*, respectively. The last column of this table presents the parameters expected from *Processor*.

Table 4.2 - Processes available in the BDC-WE.

Name	Type	Resources	Input	Output	Parameters
discovery__external	Discovery	Provider	-	List[RemoteScene]	bbox, start__date, end__date, collection, ignore__assets, limit, offset
discovery__external__by__feature	Discovery	Provider, Features	-	List[RemoteScene]	feature__id, start__date, end__date, collection, ignore__assets, limit, offset
discovery__not__processed	Discovery	Catalog	-	List[IndexedScene]	product, process__name, bbox, limit, offset
discovery__by__id	Discovery	Catalog	-	IndexedScene	id
discovery__cube	Discovery	Catalog	-	List[IndexedScene]	product, bbox, start__date, end__date
index__scene	Indexing	Catalog, Repository	LocalScene	IndexedScene	product, asset__keys
index__cube	Indexing	Catalog, Repository	LocalCube	IndexedCube	product, asset__keys
apply__scene	Processing	List[Processor], Repository	LocalScene	LocalScene	process__name, subpath, override, args
seq__apply__scene	Processing	List[Processor], Repository	LocalScene	LocalScene	process__name, subpath, override, remove__partials, args
apply__cube	Processing	List[Processor], Repository	LocalCube	LocalScube	process__name, subpath, override, args
publish__scenes	Publishing	List[Publisher], Repository	List[IndexedScene]	List[IndexedScene]	product, asset__keys
publish__cube	Publishing	List[Publisher], Repository	IndexedCube	IndexedCube	product, asset__keys

Using the *Processes* and *Resources* available in BDC-WE, the *Process Graph* illustrated in Figure 4.1 can be configured according to the diagram presented in 4.4. In this example, a STAC provider is being used as an external *RemoteScenes* provider, the *Processor* `download_proc` will be applied to each *RemoteScene* found, the `odc_-catalog` *Catalog* will be used to index the *LocalScenes* and finally, the *IndexedScenes* will be published on a Geoserver server, using the *Publisher* `geoserver_pub`. Although this *Process Graph* illustrates a flow as if only a single *RemoteScene* was found in the first *Process*, BDC-WE allows the description of *Process Graphs* that execute, for example, the *Process* `apply_scene` (`download_proc`) in parallel for each *RemoteScene* returned by *Process* `discovery_external` (`stac_provider`). More details regarding the process description of *Process Graphs* are presented in Section 4.3.

Figure 4.4 - Process graph example with resources configuration.



4.2.1.1 BDC-WE boilerplate project

To facilitate the process of deploying a BDC-WE instance, a preconfigured template project was developed to run BDC-WE with an ODC catalog. This project has a basic example of processing and a set of pre-configured services. To facilitate the deployment process, each service is run in a Docker container. These services have been grouped into four docker-compose files to make service management easier. The main file (`docker-compose.yml`) has the minimum number of services for running BDC-WE: a PostgreSQL database, a RabbitMQ messaging service, and Dagster. The worker is configured in a separate file (`docker-compose.worker.yml`) to easily run on multiple servers. The third file is `docker-compose.odc.yml`, configures the following external services: datacube-explorer[8] (STAC); Geoserver (WMS, WFS, WCS), nginx (as a file server); and a container with scripts to initialize the database and load the collections into the ODC base. The `docker-compose.openeo.yml` file configures the openEO backend and the openEO Web editor.

[8]<https://github.com/opendatacube/datacube-explorer>

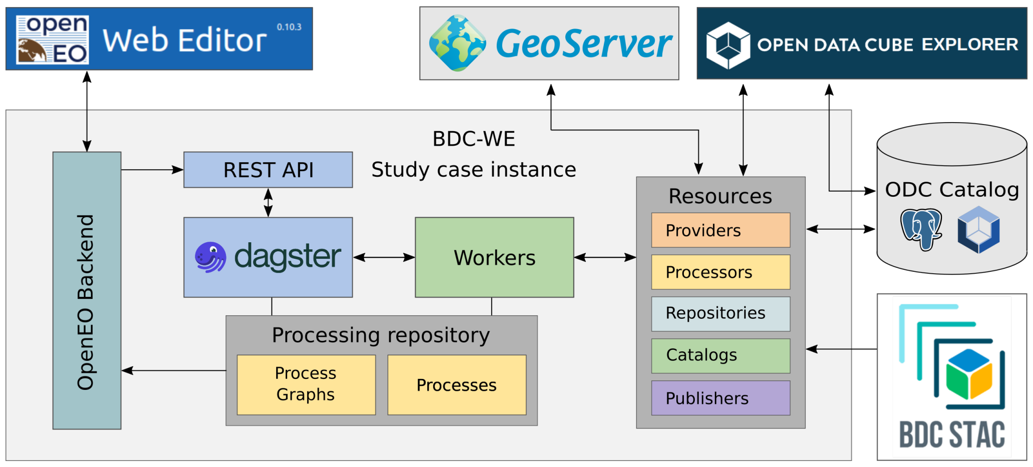
Using this complete project, it is possible to start a working BDC-WE instance. Developers can also customize this template project to meet the specific needs of an application.

4.3 Study cases

To evaluate the use of the presented BDC-WE implementation, two case studies were conducted in scenarios recurrently found in research projects at INPE. The first case study, presented in subsection 4.3.1 is more complete and deals with the operationalization of the production of water quality indices of the Mapaquali project. The second, presented in subsection 4.3.2, evaluates the use of BDC-WE for image classification using Satellite Image Time Series Analysis for Earth Observation Data Cubes (SITS) library (SIMOES et al., 2021) developed in R language. The second case study is useful for illustrating the use of the tool with an application written in a language other than Python.

The BDC-WE boilerplate project was used as the starting point in both case studies. The diagram in Figure 4.5 illustrates the services used in both the case studies. The ODC Catalog Database is a PostgreSQL instance configured with the schema used by ODC[9]. The STAC service used by Workers is the publicly available BDC[10] instance. The ODC Explorer and Geoserver services were used for data dissemination using the STAC and OGC WMS standards.

Figure 4.5 - BDC-WE study case instance diagram.



[9]<https://github.com/opendatacube/datacube-core>
[10]<https://brazildatacube.dpi.inpe.br/stac/>

4.3.1 MAPAQUALI

This case study was developed with the purpose of validating the BDC-WE in an operational environment, as a way of verifying the applicability of the system for the generation of EO data products. The MAPAQUALI project was selected for this case study. This project, being carried out at INPE's Aquatic Systems Instrumentation Laboratory (LabISA), has, among its objectives, the generation and availability of time series of the spatial distribution of water quality parameters: Chlorophyll-a, Cyanobacteria, Total Suspended Solids, Dissolved Colored Organic Matter (CDOM), an underwater light field through the diffuse attenuation coefficient (Kd), and alerts of bloom events (especially cyanobacteria). The MAPAQUALI project also demands that these water quality parameters be customized for new aquatic systems added to the platform (LABISA, 2022).

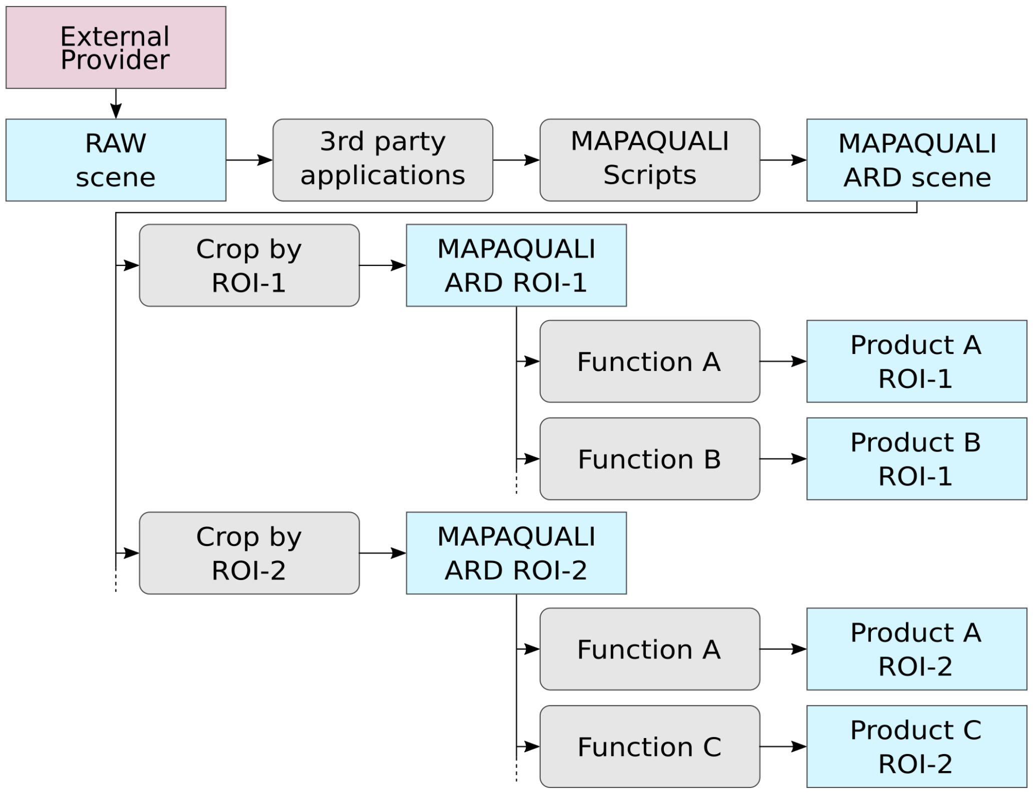
The MAPAQUALI project demands that a set of algorithms can be parameterized to generate and make available products for different areas of interest. To produce ARD, MAPAQUALI uses third-party scripts and algorithms written in Python produced by the project team. MAPAQUALI researchers also used this language to write scripts responsible for generating water quality parameter products. Most of these scripts receive the paths of the bands of a scene, the algorithm configuration parameters, and the path(s) of the file(s) of the product(s) that will be calculated as input parameters.

Figure 4.6 illustrates the general data flow of the products calculated using the MAPAQUALI project. The blue rectangles represent the data used or produced, whereas the gray rectangles illustrate the processing tasks performed.

From the collection of raw data (Landsat-8 OLI and Sentinel 2 L1C TOA) in an *External Provider*, a sequence of algorithms is applied to produce analysis-ready data (MAPAQUALI-ARD) used as a source for other products. The second stage deals with the mapping of MAPAQUALI-ARD for the regions of interest (ROI) of the project. In the case of MAPAQUALI, these ROI were lakes, water reservoirs, and other aquatic systems. In the third step, the clipped ARDs were used as inputs to the functions that produced the water quality indices developed by the LabISA research group. The same function that produces a water quality product can be used for different ROIs or be exclusive to a single ROI.

The STAC and OGC WMS services were chosen to disseminate data produced by the MAPAQUALI project. The WMS is used to view the data on the web portal

Figure 4.6 - Mapaquali products generation dataflow.



of the MAPAQUALI project, whereas the STAC allows users to consult the catalog and download the products generated by the project. As a metadata catalog, the Open Data Cube was chosen, since this framework meets the needs of the project and also provides the application datacube-explorer[11], which provides a STAC implementation and a visual interface for navigating between indexed collections in the catalog. For the WMS service, Geoserver[12] was chosen, due to the previous experience of the MAPAQUALI team in the use and configuration of this server and the availability of a resource Publisher for Geoserver implemented by BDC-WE. In addition to publishing data in the WMS service, it was decided to publish messages in the Discord communication application. Thus, the MAPAQUALI team can monitor the generation of products more easily.

Briefly, the Resources selected for use in the case study of the MAPAQUALI project were: i) stac (Provider); ii) odc (Catalog); iii) GDAL-Features (Features); and iv)

[11]<https://github.com/opendatacube/datacube-explorer>
[12]<https://geoserver.org>

odc-stac and discord (Publishers). Wrapper functions were created as Processors for each legacy function previously developed by the MAPAQUALI team. The wrapper functions are responsible for receiving the parameters in the format used by BDC-WE and passing them on to the legacy functions in the format they expect. Listing 4.1 presents an example of the recurring structure of wrapper functions used.

Listing 4.1 - Wrapper function example.

```python
def processor_a(in_scene: Scene,  dest_path: Path, resources: dict,
    **kwargs) -> LocalScene:

    prod_file = Path(dest_path, "prod_a.TIF")

    # call MAPAQUALI processing function
    function_a(prod_file, in_scene.measurement('B01').path, kwargs[
        'nodata'])

    return LocalScene(measurements=[Measurement(path=prod_file,
        properties=MeasurementProperties(name="prod_a", data_type=
        DataType.float32))])
```

Processing Graphs (PG) were created from the identified data flow to perform processing. To facilitate reading, the PG will be presented, indicating the process used and resource(s) used in parentheses. For example, `index_scene` (Catalog: odc) indicates the use of the process index_scene configured with the Open Data Cube catalog. Arrows indicate the direction of data flow.

The *PGs* used in the case study of the MAPAQUALI project are:

- **Collect Scenes**: `discovery_external` (Provider: stac) → `apply_scene` (Processor: download) → `index_scene` (Catalog: odc) → `publish` (Publishers: stac, geoserver, discord);

- **ARD**: `discovery_by_id` (Provider: odc) → `mq_scene` (Processor: download) →
`index_scene` (Catalog: odc) → `publish` (Publishers: stac, geoserver, discord);

- **ARD ROI**: `discovery_by_id` (Provider: odc) → `crop` (Processor: crop, Features: GDAL-features) → `index_scene` (Catalog: odc) → `publish` (Publishers: stac, geoserver, discord); e

- **Product A**: `discovery_by_id` (Provider: odc) → `apply_scene` (Processor: processor_a) → `index_scene` (Catalog: odc) → `publish` (Publishers: stac, geoserver, discord); e

The *Process* used are those listed in Table 4.2, while the *Resources* are those listed in Table 4.1. *Processor* mq_ard is a wrapper function that calls third-party scripts and MAPAQUALI functions to produce ARD data. *Process Graph* **Product A** represents the template used to generate the different products of the MAPAQUALI project, while processor_a refers to a wrapper function that, for example, calls a function that calculates one of the indices of water quality developed by the project's researchers.

Listing 4.2 shows how the description of PG **Product A** is performed through a JSON file using the specification of the standard JSON Graph Format[13].

Listing 4.2 - Example of Process Graph description.

```
{"graph": {
  "id": "process_a",
  "label": "Produce the water quality index A",
  "nodes": {
    "discovery_by_id": {
      "metadata": {
        "type": "discovery_by_id",
        "resources": {"provider": "odc"}}},
    "apply_scene": {
      "metadata": {
        "type": "apply_scene",
        "resources": {"processors": ["process_a"]}}},
    "index_scene": {
      "metadata": {
        "type": "index_scene",
        "resources": {"catalog": "odc"}}},
    "publish": {
      "metadata": {
        "type": "publish",
        "resources": {"publishers": ["stac","geoserver", "discord"]}}}
  },
  "edges": [
    { "source": "discovery_by_id",
      "target": "apply_scene",
      "relation": "map"},
```

[13]Available at <https://jsongraphformat.info/>

```
    { "source": "apply_scene",
      "target": "index_scene",
      "relation": "map"},
    { "source": "index_scene",
      "target": "publish",
      "relation": "collect"}
  ]}
}
```

Initially, each node of *PG* is defined. The `type` attribute indicates the *Processor* to be invoked. Optionally, it is possible to define the `resources` that are used by the *Processor*. The dependencies between nodes are defined in the `edges` attribute of the `graph`. For each dependency, the origin and destination of the data and the type of relation (`map` or `collect`) were provided. The *Resources* used in a *PG* must be previously defined in a configuration file for BDC-WE to manage these artifacts.

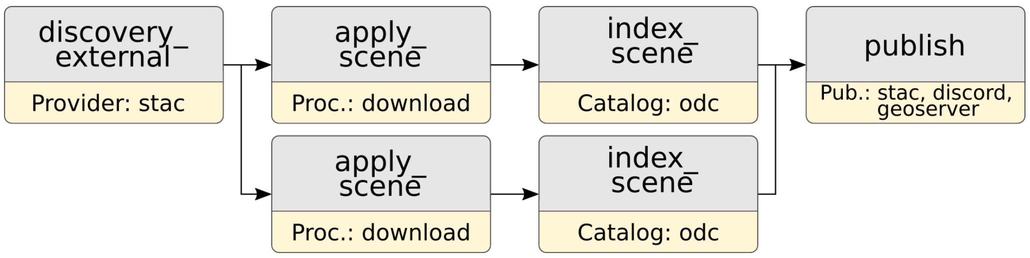
Listing 4.3 illustrates the configuration of *Process* `process_a`, which does not demand any other *Resources* and will receive, in addition to the process interface parameters defined by BDC-WE, the argument `nodata`. These arguments are useful so that processing function parameters can be configured without being previously defined in wrapper functions.

Listing 4.3 - Processors config example.

```
{
    "processors": {
        "process_a": {
            "path": "/path/to/wrappers.py",
            "function": "process_a",
            "resources": [],
            "args": { "nodata": 0.0 }
        }
        ...
    }
}
```

The configuration of data dependencies was performed using the `deps` attribute. Dependency `map:discovery_by_id` indicates that `apply_scene` is mapped (`map`) to each scene produced by `discovery_by_id`. On the other hand, the `collect:index_scene` dependency indicates that all scenes produced by `index_scene` are grouped into a list (`collect`) and then passed on to *Process* `publish`. When only one scene is produced by each *Process*, the processing flow is performed sequentially. On the other hand, if a *Process* produces a set of scenes, the next process can be performed in parallel. The concurrent execution of the *Processes* is managed by the *WO*. For example, running *PG* **Collect Scenes** that found two new scenes on the *External Provider* is illustrated by the diagram in Figure 4.7. The dependency of type `map` was used to map the multiple results of `discover_external` for each execution of `apply_scene`. The dependency of type `collect` performs grouping of results before invoking *Process* `publish`. The `map` type dependency can also be used between a *Process* that produces only one scene and one that consumes only one scene.

Figure 4.7 - Process Graph Collect Scenes diagram.



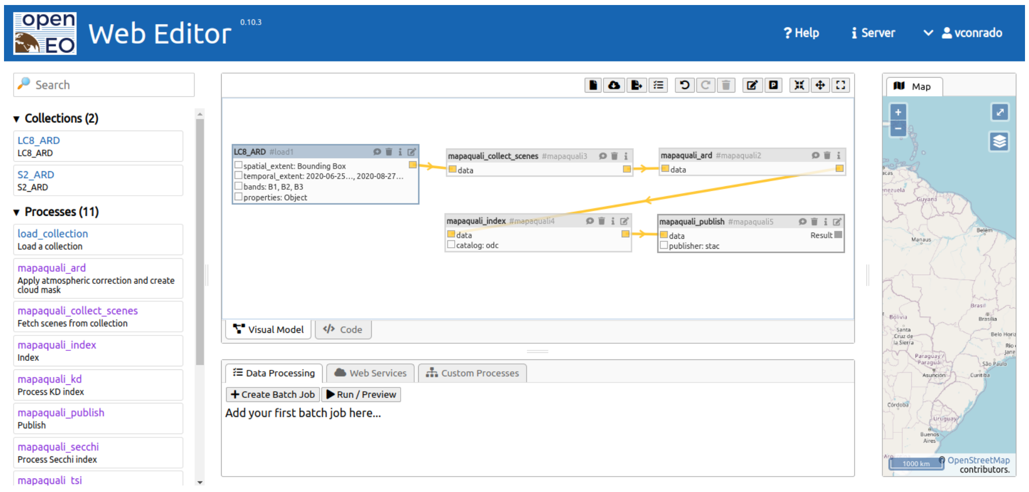
Schedulers are used for recurring execution of PGs. In the case of MAPAQUALI, this feature is used for PGs of the **Collect Scene** type, which searches for new scenes from external providers daily.

For the initialization of the other PGs, the Sensor resource was used. This functionality is used in the following situations:

- PG **ARD** begins when a new scene is downloaded by PG **Collect Scenes**.
- PG **ARD ROI** when a new scene is produced by PG **ARD**; and
- PGs of type **Product A** begin when a new scene is produced by the PG **ARD ROI**.

The PGs used in this instance of MAPAQUALI's BDC-WE were also available for execution through the OpenEO interface. In this manner, researchers can execute algorithms with different parameters to carry out tests. The generated products can be downloaded to the researcher's desktop using OpenEO's API. These products are saved in a staging repository separate from the main repository. Likewise, the metadata of these products generated by researchers via the OpenEO API are not indexed in MAPAQUALI's main catalogue. This choice was motivated to avoid contamination of research data with products made available to the public. Figure 4.8 shows the OpenEO Web Editor interface for the BDC-WE Mapaquali instance.

Figure 4.8 - OpenEO Web Editor for the BDC-WE Mapaquali instance.



4.3.2 SITS classification

SITS is an open-source R package used for satellite image time-series analysis. In the context of the BDC project, this package is used in the LULC map-generation process. The task of generating these maps is often divided into two phases: training and classification. In the training phase, previously classified samples from a region of interest (ROI) are used to calibrate the predictive model. With this model, the scenes of this ROI, previously structured in the form of a data cube by the SITS package, are then classified.

For the current case study, a *Process Graph* was created to classify the SITS data cube, considering the existence of a previously calibrated predictive model. This classification phase consists of the following function invocation sequence from the SITS package:

a) `sits_cube`: performs the query on the BDC STAC and defines one of a data cube for the region of interest;

b) `sits_classify`: performs scene classification using a predictive model and the data cube produced in the previous step;

c) `sits_smooth`: performs the smoothing of the classification performed in the previous step; and

d) `sits_label_classification`: from the scene probability values, it converts to a label based on the highest probability of each pixel.

This algorithm was divided into two phases to run on BDC-WE, considering a *PG* type *Discovery* -> *Process* -> *Index* -> *Publish*. In the search phase, a new Provider, SitsProvider, was implemented. SitsProvider's search method invokes the sits_cube function and produces a list of scenes to be processed. For this, a script was created in the R language, which receives the necessary parameters and invokes the `sits_cube` function. The data cube definition resulting from this function is then spatially split to define the smaller data cubes. This subdivision is performed by considering the grid used by the BDC for the collection. The purpose of this split was to make it easier to parallelize the sort run on each data cube. The data structure in R, representing the metadata of these smaller data cubes, was saved in `.rda` format. The `search` method of SitsProvider returns a list of objects of type SitsRemoteScene (which extends the RemoteScene class). Each of these objects has among its attributes the path to one of the data cubes generated by the script in R. This approach was used to represent an object produced in Python and processed in the R language.

The objects produced in the previous phase (discovery) were passed to a classified processor. This function follows the structure presented in listing 4.1, and calls a script in R called classify.R, which receives as input parameter the path of the predictive model and file *.rda* of the data cube. This script is responsible for performing classification, smoothing, and labeling of the pixels of each data cube. In addition, it returned the path of the sorted file. This path is for the processor to classify and create the scene object, which is returned to the Process Graph.

The classified scenes were then indexed into an STAC catalog and published to an OGC WMS service (Geoserver) and an STAC catalog. Figure 4.9 illustrates the Process Graph used in the case study and Figure 4.10 shows the results of the classification performed in this study.

Figure 4.9 - *Process Graph* diagram for image classification with SITS.

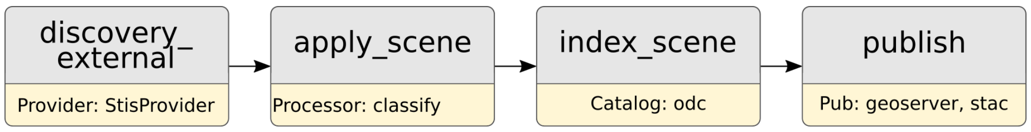
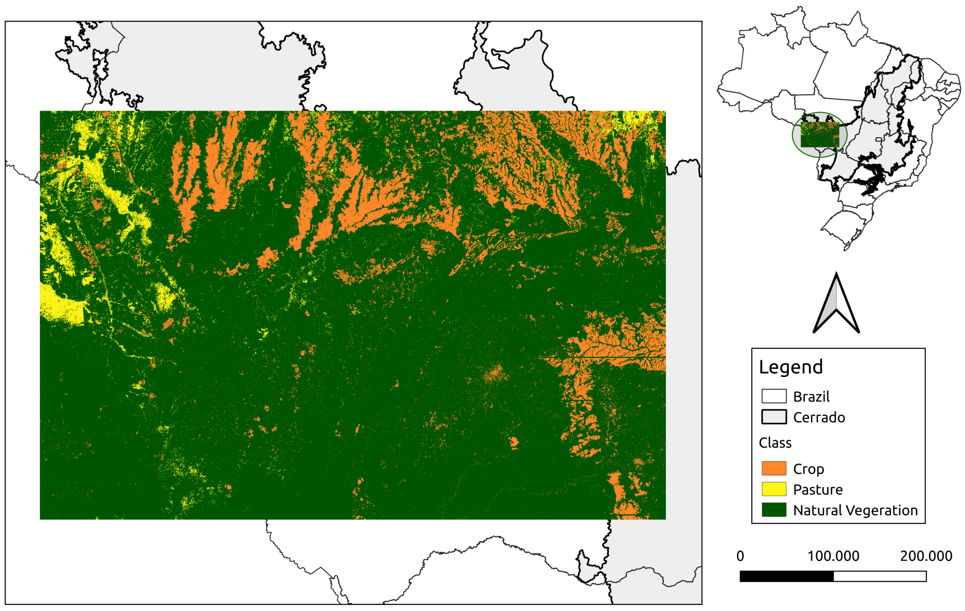


Figure 4.10 - Western region of the cerrado biome classified using the SITS package running on the BDC-WE.



4.4 Final remarks and discussion

This study presents an architecture for the processing of workflows for processing EO data. This architecture addresses the possibility of flexibility for the inclusion of new algorithms while also providing a high-level interface for users, namely, the OpenEO API. A prototype was developed and evaluated using two cases.

The case study of the Mapaquali project showed us that the use of *Process* that represent meta-tasks made the process of creating PGs easier with algorithms previously developed by the Mapaquali team. Through wrapper functions and the configuration of a *PG* with four *Process* (Discovery, Process, Index, and Publish), it was possible to produce most of the products of this project. Dagster's scheduling functionality was useful in this use case, as it allows new scenes to be found daily from providers and processed by the configured *PG*. Currently, the Mapaquali project has been using an exclusive instance of the BDC-WE prototype to produce water quality indices provided by the project.

Regarding the second use case, we observed that the decomposition of the algorithm for classifying images into subtasks and the description through a PG facilitates the processing of massive sets of data for the production of classification maps using SITS. The OpenEO API can be used by users and developers to select a region of interest and provide a file with a model previously trained by SITS to the BDC-WE, which executes this classification PG distributing the *Process* to all available *Workers*.

These two case studies show us that the BDC-WE allowed applications, which were initially implemented to be executed sequentially or in parallel on a single machine, to easily gain processing scale. This is possible because of the description of these applications in *Processes*, which can be orchestrated by the BDC-WE. In this manner, once the application is modeled in the form of *PG*, new computational resources can be accommodated in the cluster to allow a gain in the processing scale, without the need for any change in *PG*.

The integration of OpenEO with *WO* through requests to the GraphQL API proved to be efficient because it is possible to access all the resources available in Dasgster. This separation between *WO* and the module responsible for processing requests allows, for example, new APIs to be integrated or developed in the future without the need to change the way processing is carried out. The advantage of using OpenEO as a high-level interface for BDC-WE is the availability of tools, such as the OpenEO Web Editor and clients in three different programming languages, and the possibility of future integration with other EO data processing platforms using this API.

Using a development-ready OpenEO Backend framework accelerated the integration of this API into BDC-WE. In particular, to make collections available, calls were made to methods already implemented by a *Resource* of type *Catalog*. However, the implementation of the entire set of operators available in this API requires considerable effort. In the case of the BDC-WE prototype, in which the purpose was to validate the proposed architecture, a reduced set of operators was implemented, such as cross-band operation, time reduction (mean, maximum, minimum, and standard deviation), and invocation of pre-defined *PG* specific to each use case.

The results obtained until the present moment of development of this research motivates us to establish a continuity. Among the points that we would like to address in future work, we highlight the individualized management of the use of computational resources, reproducibility, and code sharing. Regarding the management of the use of computational resources, OpenEO API defines endpoints for billing management, such as checking the credit available to the user, cost estimate for operations, and information on the costs of operations performed. However, this API does not define how these operations should be performed. In the BDC-WE architecture, the BDC-WE REST-API module that mediates all processing requests is responsible for these activities. A possible solution to this issue is the use of an approach inspired by the solution adopted by GEE, which limits the amount of RAM and CPU memory per processing. In the case of BDC-WE, the expectation is to limit the use of RAM by *Process* and use CPU time as a metric to be discounted from users' credits. The limits of memory and CPU used by a *Process* can be established in the execution of Docker containers and technology currently in use by BDC-WE.

In relation of code sharing, although the use of the OpenEO API facilitates this process in BDC-WE, it is still up to the researchers to manage the exchange of files among their peers. The ability to share the analyses is the first step in the path to reproducibility. Carlos (2022)'s work presents a tool to assist in the process of managing research artifacts to ensure reproducible sharing, and should be considered as an important source of inspiration for including this capability in the BDC-WE.

In addition to these works, we intend to move forward with the implementation of BDC-WE through the implementation of all operators available in the OpenEO API, and automate the loading and availability of *PGs* through configuration files. As the implementation of this tool advances, our goal is to make BCD-WE the central tool for carrying out processing on the BDC platform, being responsible for both processing user analyses and executing platform-specific applications, such as the *Collection Builder* and *Cube Builder*.

5 FINAL REMARKS

In this work, a set of activities was carried out that can be grouped into three phases, which were presented in Chapters 2, 3 and 4.

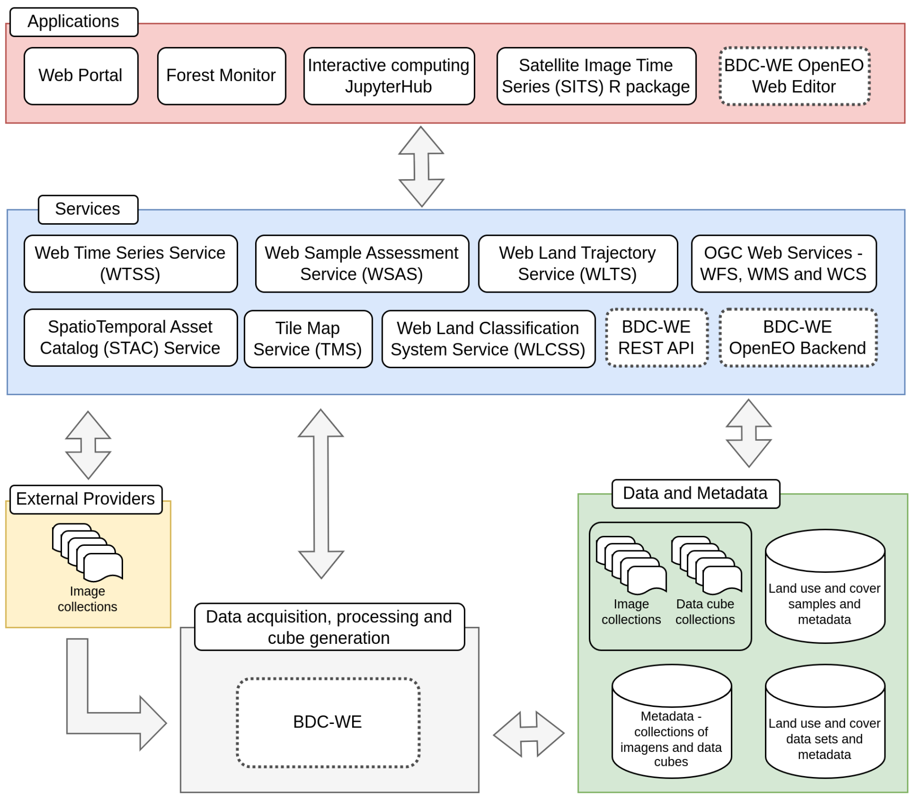
In the first stage, a study and exploration of the main platforms for processing big EO data were carried out. Each platform was evaluated considering ten capabilities, and a comparative analysis was performed. It is noteworthy that, in this study, the objective is not to suggest that an "ideal" platform would have all the aforementioned capabilities at a full level. This theme is brought to light so that users and developers can assess which features might best meet their requirements.

In the second stage, a set of technologies was explored, and the ODC framework was chosen to join the other services available on the BDC platform. The integration of different big EO data platforms can provide gains for the users of both platforms. In the work carried out, users familiar with ODC had access to a new dataset and environment for processing, whereas BDC users benefited from a large catalog of algorithms available for the ODC framework.

From the knowledge acquired in the previous phases, we observed that the main open-source platforms and technologies provide tools for data indexing (ODC) or high-level interfaces for processing descriptions (OpenEO). SEPAL, on the other hand, focuses on providing a private cloud processing environment for researchers with restricted access to these resources. The solutions that offer more complete processing capabilities are focused on running on-premise servers such as GEE, JEODPP, and pipsCloud. This choice is mainly owing to the need for these solutions to be integrated with the available hardware and software resources and the needs of the target users of each platform. Decisions that consider specific needs add value to platform users, which often makes them unfeasible for users with specific applications.

In the third stage, the architecture of a tool called BDC-WE is proposed for processing big EO data using the concept of workflows. A prototype of this tool was implemented and two use cases were realized. The focus of this case is to fill the gap in terms of structuring and orchestrating large-scale processing in BDC platform. Metadata cataloging and high-level interfaces are already more established and available, such as, respectively, ODC and OpenEO. Thus, we use these two technologies in the implemented prototype. In the BDC platform scenario, BDC-WE represents part of the data processing solutions. Figure 5.1 presents an updated diagram with the BDC-WE tool proposed in this work together with the other software and data products of the Brazil Data Cube platform. In this diagram, the elements highlighted by the dotted border represent the BDC-WE modules, developed in the context of this thesis.

Figure 5.1 - Updated diagram of software and data products of Brazil Data Cube Project (FERREIRA et al., 2020) integrated with the BDC-WE tool.


Applications
Web Portal
Forest Monitor
Interactive computing JupyterHub
Satellite Image Time Series (SITS) R package
BDC-WE OpenEO Web Editor
Services
Web Time Series Service (WTSS)
Web Sample Assessment Service (WSAS)
Web Land Trajectory Service (WLTS)
OGC Web Services - WFS, WMS and WCS
SpatioTemporal Asset Catalog (STAC) Service
Tile Map Service (TMS)
Web Land Classification System Service (WLCSS)
BDC-WE REST API
BDC-WE OpenEO Backend
External Providers
Image collections
Data acquisition, processing and cube generation
BDC-WE
Data and Metadata
Image collections
Data cube collections
Land use and cover samples and metadata
Metadata - collections of imagens and data cubes
Land use and cover data sets and metadata

In this Figure, the BDC-WE tool is illustrated as the primary tool for the data acquisition and processing, and generation of data cubes. The BDC-WE APIs (BDC-WE REST API and BDC-WE OpenEO API) are a part of the service layer of the BDC platform. The OpenEO Web Editor, which also indirectly represents OpenEO libraries in R, Python, and JavaScript languages, is present in the application layer.

In synchronous applications, such as WTSS, WSAS, and WLTS, which perform minimal processing before delivering data to users, we understand that processing must continue to be performed in the context of each application. However, applications that demand more intensive processing must be executed in a coordinated execution environment, such as BDC-WE. The *Collection Builder* and *Data Builder* applications are examples of applications with these characteristics.

For the task of processing and analyzing time series using SITS, for example, we understand that the researcher will only use BDC-WE when the methodology is already consolidated. Until then, the JupyterHub environment can be used iteratively to perform sample selection, calibrate and train models, and perform test classifications. Once a model is ready, BDC-WE can be used through the OpenEO API to perform the classification of a large volume of data in a distributed environment.

The path taken thus far indicates that BDC-WE is a promising tool for definitive incorporation into the BDC platform and that it will bring significant gains to platform users. As mentioned in Section 4.4, our next implementation steps include the other operators available in the OpenEO API, the migration of the *Collection Builder* and *Data Builder* applications to run on the BDC -WE and the availability of the OpenEO client in the JupyterHub environment of the BDC platform. In the field of research, our focus will turn to issues of accountability of resources used by users, reproducibility, and code sharing.

REFERENCES

ADDE, G.; CHAN, B.; DUELLMANN, D.; ESPINAL, X.; FIOROT, A.; IVEN, J.; JANYST, L.; LAMANNA, M.; MASCETTI, L.; ROCHA, J. M.; PETERS, A. J.; SINDRILARU, E. A. Latest evolution of EOS filesystem. **Journal of Physics: Conference Series**, v. 608, n. 1, 2015. 25

AMAZON WEB SERVICES. **Open data on AWS**. 2020. Available from: <https://aws.amazon.com/opendata/>. Access on: 26 Mar. 2020. 13

APPEL, M.; LAHN, F.; BUYTAERT, W.; PEBESMA, E. Open and scalable analytics of large Earth observation datasets: from scenes to multidimensional arrays using SciDB and GDAL. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 138, p. 47–56, apr 2018. ISSN 0924-2716. 1, 11

APPEL, M.; PEBESMA, E. On-demand processing of data cubes from satellite image collections with the gdalcubes library. **Data**, v. 4, n. 3, p. 92, 2019. ISSN 2306-5729. 1, 41, 45, 53

ARIZA-PORRAS, C.; BRAVO, G.; VILLAMIZAR, M.; MORENO, A.; CASTRO, H.; GALINDO, G.; CABERA, E.; VALBUENA, S.; LOZANO, P. CDCol: a geoscience data cube that meets colombian needs. In: **Solano, A., Ordoñez, H. (eds) Advances in computing**. Cham: Springer, 2017. p. 87–99. 8, 23, 32, 33, 45

ASMARYAN, S.; MURADYAN, V.; TEPANOSYAN, G.; HOVSEPYAN, A.; SAGHATELYAN, A.; ASTSATRYAN, H.; GRIGORYAN, H.; ABRAHAMYAN, R.; GUIGOZ, Y.; GIULIANI, G. Paving the way towards an armenian data cube. **Data**, v. 4, n. 3, p. 11, 2019. ISSN 2306-5729. 45

ASSIS, L. F. F. G. d.; QUEIROZ, G. R. de; FERREIRA, K. R.; VINHAS, L.; LLAPA, E.; SANCHEZ, A. I.; MAUS, V.; CâMARA, G. Big data streaming for remote sensing time series analytics using mapreduce. **Revista Brasileira de Cartografia**, v. 69, n. 5, 2017. 13

BAUMANN, P. The OGC web coverage processing service (WCPS) standard. **GeoInformatica**, v. 14, n. 4, p. 447–479, 2010. ISSN 13846175. 13

BAUMANN, P.; DEHMEL, A.; FURTADO, P.; RITSCH, R.; WIDMANN, N. The multidimensional database system RasDaMan. **ACM SIGMOD Record**, v. 27, n. 2, p. 575–577, 1998. ISSN 01635808. 12

BAUMANN, P.; MAZZETTI, P.; UNGAR, J.; BARBERA, R.; BARBONI, D.; BECCATI, A.; BIGAGLI, L.; BOLDRINI, E.; BRUNO, R.; CALANDUCCI, A.; CAMPALANI, P.; CLEMENTS, O.; DUMITRU, A.; GRANT, M.; HERZIG, P.; KAKALETRIS, G.; LAXTON, J.; KOLTSIDA, P.; LIPSKOCH, K.; MAHDIRAJI, A. R.; MANTOVANI, S.; MERTICARIU, V.; MESSINA, A.; MISEV, D.; NATALI, S.; NATIVI, S.; OOSTHOEK, J.; PAPPALARDO, M.; PASSMORE, J.; ROSSI, A. P.; RUNDO, F.; SEN, M.; SORBERA, V.; SULLIVAN, D.; TORRISI, M.; TROVATO, L.; VERATELLI, M. G.; WAGNER, S. Big data analytics for earth sciences: the EarthServer approach. **International Journal of Digital Earth**, v. 9, n. 1, p. 3–29, 2016. 13

BLOMER, J. A survey on distributed file system technology. **Journal of Physics**, 2014. 13

BRAZIL DATA CUBE PROJECT. **Brazil Data Cube platform**. 2022. Available from: <http://www.brazildatacube.org/>. Access on: 20 July 2022. 60

BROWN, M. E. Remote sensing technology and land use analysis in food security assessment. **Journal of Land Use Science**, v. 11, n. 6, p. 623–641, 2016. 59

CAMARA, G.; ASSIS, L. F.; RIBEIRO, G.; FERREIRA, K. R.; LLAPA, E.; VINHAS, L. Big earth observation data analytics: matching requirements to system architectures. In: ACM SIGSPATIAL INTERNATIONAL WORKSHOP ON ANALYTICS FOR BIG GEOSPATIAL DATA, 5., 2016, California, USA. **Proceedings...** New York: ACM, 2016. p. 1–6. 8, 11, 13, 32, 33, 43, 59

CAMARA, G.; EGENHOFER, M. J.; FERREIRA, K.; ANDRADE, P.; QUEIROZ, G.; SANCHEZ, A.; JONES, J.; VINHAS, L. Fields as a generic data type for big spatial data. **Geographic information science**, p. in press, 2014. ISSN 16113349. 1

CAMARA, G.; SIMOES, R.; ANDRADE, P. R.; MAUS, V.; SáNCHEZ, A.; ASSIS, L. F. F. G. de; SANTOS, L. A.; YWATA, A. C.; MACIEL, A. M.; VINHAS, L.; QUEIROZ, G. **e-sensing/sits: version 1.12.5**. [S.l.]: Zenodo: Geneva, Switzerland, 2018. Available from: <https://doi.org/10.5281/zenodo.1974065>. 44, 53

CARLOS, F. M. **Storm: platform to support the development of reproducible and collaborative geospatial applications**. Dissertation (Master in Applied Computing) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2022. 85

CEOS. **CEOS repository**. 2021. Available from: <https://github.com/ceos-seo/>. Access on: 25 Jan. 2021. 46, 52, 55

COPERNICUS. **DIAS | Copernicus**. 2020. Available from: <https://www.copernicus.eu/en/access-data/dias/>. Access on: 26 Mar. 2020. 13

CREODIAS. **What is CREODIAS?** 2020. Available from: <https://creodias.eu/>. Access on: 26 Mar. 2020. 13, 14

ELEMENTL. **Dagster - Cloud-native orchestration of data pipelines**. 2022. Available from: <https://dagster.io/>. Access on: 20 July 2022. 68

EUROPEAN SPACE AGENCY. **openEO platform**. 2022. Available from: <https://openeo.cloud/>. Access on: 20 July 2022. 2, 60

FERREIRA, K. R.; QUEIROZ, G. R.; MARUJO, R. F. B.; COSTA, R. W. Building earth observation data cubes on AWS. **The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, XLIII-B3-2022, p. 597–602, 2022. 61

FERREIRA, K. R.; QUEIROZ, G. R.; VINHAS, L.; MARUJO, R. F.; SIMOES, R. E.; PICOLI, M. C.; CAMARA, G.; CARTAXO, R.; GOMES, V. C.; 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.; FONSECA, L. M. Earth observation data cubes for Brazil: requirements, methodology and products. **Remote Sensing**, v. 12, n. 24, p. 1–19, 2020. ISSN 20724292. xiv, 1, 44, 45, 46, 48, 51, 61, 88

FOOD AND AGRICULTURE ORGANIZATION (FAO). **SEPAL repository**. 2020. Available from: <https://github.com/openforis/sepal/>. Access on: 07 Feb. 2020. 14, 23, 24

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-t. The Google file system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 2003, Bolton Landing, New York, USA. **Proceedings...** New York, NY, USA: ACM, 2003. p. 20–43. 12

GIULIANI, G.; CAMARA, G.; KILLOUGH, B.; MINCHIN, S. Earth observation open science: enhancing reproducible science using data cubes. **Data**, v. 4, n. 4, p. 4–9, 2019. ISSN 23065729. 45

GIULIANI, G.; CHATENOUX, B.; De Bono, A.; 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**, v. 1, n. 1-2, p. 100–117, 2017. ISSN 2096-4471. 23, 40, 45

GIULIANI, G.; MASÓ, J.; MAZZETTI, P.; NATIVI, S.; ZABALA, A. Paving the way to increased interoperability of Earth Observations Data Cubes. **Data**, v. 4, n. 3, p. 23, 2019. 43, 44, 58

GOLDBLATT, R.; YOU, W.; HANSON, G.; KHANDELWAL, A. Detecting the boundaries of urban areas in India: a dataset for pixel-based image classification in Google Earth Engine. **Remote Sensing**, v. 8, n. 8, p. 634, aug 2016. ISSN 2072-4292. 18

GOMES, V. C.; CARLOS, F. M.; QUEIROZ, G. R.; FERREIRA, K. R.; SANTOS, R. Accessing and processing Brazilian Earth Observation Data Cubes with the Open Data Cube platform. **ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, v. 5, n. 4, p. 153–159, 2021. ISSN 21949050. 43, 61, 62

GOMES, V. C. F.; QUEIROZ, G. R.; FERREIRA, K. R. An overview of platforms for big earth observation data management and analysis. **Remote Sensing**, v. 12, n. 8, p. 25, 2020. 1, 2, 11, 44, 60, 62, 67

GOOGLE. **Google Earth Engine**. 2020. Available from: <https://earthengine.google.com/>. Access on: 27 Mar. 2020. 1, 18

GORELICK, N.; HANCHER, M.; DIXON, M.; ILYUSHCHENKO, S.; THAU, D.; MOORE, R. Google Earth Engine: planetary-scale geospatial analysis for everyone. **Remote Sensing of Environment**, v. 202, n. 2016, p. 18–27, 2017. ISSN 00344257. 14, 15, 16, 17, 44, 60

GUO, Z.; FOX, G.; ZHOU, M. Investigation of data locality in MapReduce. In: INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING, 12., 2012, Ottawa, ON, Canada. **Proceedings...** Ottawa: IEEE/ACM, 2012. p. 419–426. 12, 13

KILLOUGH, B. Overview of the open data cube initiative. In: INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM (IGARSS), 2018, Valencia, Spain. **Proceedings...** Valencia: IEEE, 2018. p. 8629–8632. 44, 45, 56

_____. The impact of analysis ready data in the Africa Regional Data Cube. In: INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM, Yokohama, Japan. **Proceedings...** Yokohama: IEEE, 2019. p. 5646–5649. ISBN 978-1-5386-9154-0. 41

LABISA. **Mapaquali**. 2022. Available from: <http://www.dpi.inpe.br/labisa/project/mapaquali/>. Access on: 20 July 2022. 75

LEWIS, A.; OLIVER, S.; LYMBURNER, L.; EVANS, B.; WYBORN, L.; MUELLER, N.; RAEVKSI, G.; HOOKE, J.; WOODCOCK, R.; SIXSMITH, J.; WU, W.; TAN, P.; LI, F.; KILLOUGH, B.; MINCHIN, S.; ROBERTS, D.; AYERS, D.; BALA, B.; DWYER, J.; DEKKER, A.; DHU, T.; HICKS, A.; IP, A.; PURSS, M.; RICHARDS, C.; SAGAR, S.; TRENHAM, C.; WANG, P.; WANG, L. W. The Australian Geoscience Data Cube — foundations and lessons learned. **Remote Sensing of Environment**, v. 202, p. 276–292, 2017. ISSN 00344257. 21, 23, 40, 45

MARUJO, R. F. B.; FERREIRA, K. R.; QUEIROZ, G. R.; COSTA, R. W.; ARCANJO, J. S.; SOUZA, R. C. M. Generating analysis ready data collections for Brazil. In: INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM, 2022, Kuala Lumpur, Malaysia. **Proceedings...** Kuala Lumpur: IEEE, 2022. p. 6844–6847. 61

MERTICARIU, G.; MISEV, D.; BAUMANN, P. Towards a general array database benchmark: measuring storage access. In: **Rabl, T., Nambiar, R., Baru, C., Bhandarkar, M., Poess, M., Pyne, S. (eds) Big data benchmarking**. [S.l.]: Springer, 2015. p. 40–67. 12

MÜLLER, M. **Service-oriented geoprocessing in Spatial Data Infrastructures**. 123 p. Thesis (PhD in Natural Sciences) — Technische Universität Dresden, Dresden, 2016. 1, 12, 13, 43, 59, 62

MÜLLER, M.; BERNARD, L.; BRAUNER, J. Moving code in spatial data infrastructures - web service based deployment of geoprocessing algorithms. **Transactions in GIS**, v. 14, n. Suppl. 1, p. 101–118, 2010. ISSN 13611682. 11, 59

MUNDI WEB SERVICES. **Mundi Web Services**. 2020. Available from: <https://mundiwebservices.com/>. Access on: 26 Mar. 2020. 13, 14

NATIVI, S.; MAZZETTI, P.; CRAGLIA, M. A view-based model of data-cube to support big earth data systems interoperability. **Big Earth Data**, v. 1, n. 1-2, p. 75–99, dec 2017. ISSN 2096-4471, 2574-5417. 41, 43, 44

ONDA. **ONDA**. 2020. Available from: <https://www.onda-dias.eu/>. Access on: 26 Mar. 2020. 13, 14

OPEN DATA CUBE. **The "Road to 20" international data cube deployments**. [S.l.], 2019. 10 p. 23, 44

_____. **Open Data Cube repository**. 2020. Available from: <https://github.com/opendatacube/>. Access on: 07 Feb. 2020. 23

_____. **Open Data Cube manual**. 2021. Available from: <https://datacube-core.readthedocs.io/en/latest/>. Access on: 25 Jan. 2021. 22, 46

_____. **ODC stats repository**. 2022. Available from: <https://github.com/opendatacube/odc-stats>. Access on: 20 July 2022. 62

_____. **Open Data Cube**. 2022. Available from: <https://www.opendatacube.org/>. Access on: 13 Jun. 2022. 1, 14, 22, 44, 60

OPEN GEOSPATIAL CONSORTIUM (OGC). **OGC standards and supporting documents**. 2019. Available from: <http://www.opengeospatial.org/standards/>. Access on: 12 Dec. 2019. 11

OPENEO. **openEO - concepts and API reference**. 2018. Available from: <https://open-eo.github.io/openeo-api/arch/index.html>. Access on: 10 Jan. 2020. 29, 31

OPENEO. **openEO documentation**. 2022. Available from: <https://api.openeo.org/>. Access on: 20 July 2022. 1, 29, 31, 63

PAGANINI, M.; PETITEVILLE, I.; WARD, S.; DYKE, G.; STEVENTON, M.; HARRY, J.; KERBLAT, F. **Satellite Earth observations in support of the sustainable development goals - Special Edition 2018**. 2022. Available from: <http://eohandbook.com/sdg/files/CEOS_EOHB_2018_SDG.pdf>. Access on: 20 July 2022. 59

PAPADOPOULOS, S.; MADDEN, S.; MATTSON, T. The TileDB array data storage manager. **Proceedings of the VLDB Endowment**, v. 10, n. i, p. 349–360, 2016. ISSN 21508097. 12

PEBESMA, E.; WAGNER, W.; SCHRAMM, M.; Von Beringe, A.; PAULIK, C.; NETELER, M.; REICHE, J.; VERBESSELT, J.; DRIES, J.; GOOR, E.; MISTELBAUER, T.; BRIESE, C.; NOTARNICOLA, C.; MONSORNO, R.; MARIN, C.; JACOB, A.; KEMPENEERS, P.; SOILLE, P. **openEO - a common, open source interface between earth observation data infrastructures and front-end applications**. [S.l.], 2017. 57 p. 14, 30

RAJABIFARD, A.; WILLIAMSON, I. P. Spatial data infrastructures: concept, SDI hierarchy and future directions. In: GEOMATICS CONFERENCE, 2001, Australia. **Proceedings...** Australia, 2001. 11, 43, 59

SEDONA, R.; CAVALLARO, G.; JITSEV, J.; STRUBE, A.; RIEDEL, M.; BENEDIKTSSON, J. A. Remote sensing big data classification with high performance distributed deep learning. **Remote Sensing**, v. 11, n. 24, p. 1–19, 2019. ISSN 20724292. 12

SHELESTOV, A.; LAVRENIUK, M.; KUSSUL, N.; NOVIKOV, A.; SKAKUN, S. Exploring Google Earth Engine platform for big data processing: classification of multi-temporal satellite imagery for crop mapping. **Frontiers in Earth Science**, v. 5, p. 1–10, 2017. ISSN 2296-6463. 17, 18

SHVACHKO, K.; KUANG, H.; RADIA, S.; CHANSLER YAHOO, R. The Hadoop distributed file system. In: SYMPOSIUM ON MASS STORAGE SYSTEMS AND TECHNOLOGIES (MSST), 26., 2010, California, USA. **Proceedings...** California: IEEE, 2010. p. 1–10. ISBN 9781424471539. 12

SIMOES, R.; CAMARA, G.; QUEIROZ, G.; SOUZA, F.; ANDRADE, P. R.; SANTOS, L.; CARVALHO, A.; FERREIRA, K. Satellite image time series analysis for big earth observation data. **Remote Sensing**, v. 13, n. 13, 2021. ISSN 2072-4292. 61, 74

SINERGISE. **Sentinel-Hub by Sinergise**. 2020. Available from: <https://www.sentinel-hub.com/>. Access on: 10 Jan. 2020. 1, 14, 18, 20, 44, 60

_____. **Sentinel-Hub documentation**. 2020. Available from: <https://docs.sentinel-hub.com/api/>. Access on: 10 Jan. 2020. 20

SOBLOO. **sobloo**. 2020. Available from: <https://sobloo.eu/>. Access on: 26 Mar. 2020. 13, 14

SOILLE, P.; BURGER, A.; DE MARCHI, D.; KEMPENEERS, P.; RODRIGUEZ, D.; SYRRIS, V.; VASILEV, V. A versatile data-intensive computing platform for information retrieval from big geospatial data. **Future Generation Computer Systems**, v. 81, p. 30–40, 2018. ISSN 0167739X. 11, 14, 26, 27, 43, 44, 60

STONEBRAKER, M.; DUGGAN, J.; BATTLE, L.; PAPAEMMANOUIL, O. SciDB DBMS research at M.I.T. **IEEE Data Engineering Bulletin**, v. 36, n. 4, p. 21–30, 2013. 12

STROMANN, O.; NASCETTI, A.; YOUSIF, O.; BAN, Y. Dimensionality reduction and feature selection for object-based land cover classification based on Sentinel-1 and Sentinel-2 time series using Google Earth Engine. **Remote Sensing**, v. 12, n. 1, 2020. ISSN 20724292. 11, 43, 59

TEMPORAL TECHNOLOGIES. **Temporal - open source durable execution platform**. 2022. Available from: <https://temporal.io/>. Access on: 20 July 2022. 68

VINHAS, L.; QUEIROZ, G. R.; FERREIRA, K. R.; CAMARA, G. Web Services for big earth observation data. In: GEOINFO, 17., 2016, Campos do Jordão, Brazil. **Proceedings...** Campos do Jordão, 2016. p. 166–177. 13, 47

WANG, L.; MA, Y.; YAN, J.; CHANG, V.; ZOMAYA, A. Y. pipscloud: high performance cloud computing for remote sensing big data management and processing. **Future Generation Computer Systems**, v. 78, p. 353 – 368, 2018. ISSN 0167-739X. 14, 27, 28, 60

WANG, W.; YING, L. Data locality in MapReduce: a network perspective. **Performance Evaluation**, v. 96, p. 1–11, 2016. ISSN 01665316. 13

WEKEO. **WEkEO**. 2020. Available from: <https://www.wekeo.eu/>. Access on: 26 Mar. 2020. 13, 14

WOODCOCK, R.; CECERE, T.; MITCHELL, A.; KILLOUGH, B.; DYKE, G.; ROSS, J.; ALBANI, M.; WARD, S.; LABAHN, S. **CEOS future data access and analysis architectures study**. [S.l.], 2016. 33 p. 12, 13, 43, 60

WU, Y.; XIANG, Y.; GE, J.; MULLER, P. High-performance computing for big data processing. **Future Generation Computer Systems**, v. 88, p. 693–695, 2018. ISSN 0167739X. 13

YUE, P.; RAMACHANDRAN, R.; BAUMANN, P.; KHALSA, S. J. S.; DENG, M.; JIANG, L. Recent activities in earth data science [technical committees]. **IEEE Geoscience and Remote Sensing Magazine**, v. 4, n. 4, p. 84–89, 2016. 12

YUE, P.; ZHANG, C.; ZHANG, M.; ZHAI, X.; JIANG, L. An SDI approach for big data analytics: the case on sensor web event detection and geoprocessing workflow. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 8, n. 10, p. 4720–4728, 2015. 12