universität wien

MASTERARBEIT / MASTER’S THESIS

Titel der Masterarbeit / Title of the Master‘s Thesis

Performance Evaluation of Sentinel-2 Earth Observation Data Cube Generation in the Context of the EU Common Agricultural Policy

verfasst von / submitted by

Stefan Brand BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 856
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Kartographie und Geoinformation
Betreut von / Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Erklärung

Hiermit versichere ich,

- dass ich die vorliegende Masterarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubter Hilfe bedient habe,
- dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe
- und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit vollständig übereinstimmt.

Wien, am……..

Acknowledgments

I would like to express my thanks to a number of people, who helped make this master's thesis possible. First of all, I would like to thank my sponsor EOX IT Services for giving me the opportunity to write the thesis in a company environment. In particular, CEO Dr. Gerhard Triebnigg put his trust in me from the beginning and Stefan Achtsnit and Joachim Ungar supported me with discussion about the research and by answering all my questions.

My supervisor Ass.-Prof. Andreas Riedl enabled me to finish the thesis quickly by always replying promptly to my e-mails. Besides, his concise feedback encouraged me to dig deeper into the research topic and perform meticulous scientific writing.

During the whole process of writing this master's thesis my mother took a lot of load off me. Thank you for always being there for me and supporting my decisions! Unfortunately I have not won the race between my girlfriend and me, about who would first finish their studies. ;-) Nevertheless I want to thank you for pushing me forward and keeping my self-set deadline in my mind, leading to continuous research and writing progress. I love you both!

Lastly, but most importantly, I want to thank my Heavenly Father and His Son Jesus Christ for blessing my life and putting me in a position where I can live and learn freely.

Abstract

Among the EU policies the Common Agricultural Policy (CAP) has one of the biggest budget shares, most of which is distributed to farmers. In 2018 the member states' paying agencies disbursed about 42 billion Euros of direct payments. In return the farmers commit to environmentally friendly farming practices and landscape conservation. (EU, 2020c) Compliance is monitored via On-The-Spot-Checks (OTSC), which is time-consuming and only allows for 5 % of farms being inspected. (Devos, Fasbender, Griffiths, et al., 2017, p. 4)

This is where the Copernicus Program with its Sentinel-1 and -2 satellites comes in. In an effort to modernize the CAP and improve its efficiency the EU wants to move to Checks by monitoring (CbM) and do automatic compliance inspections of all agricultural parcels. CbM is facilitated by training a Machine Learning (ML) model using labelled satellite data. The ML model then classifies the agricultural parcels according to crop type and compares this computed crop type to the one declared by the farmer. Non-complying parcels are highlighted to the inspectors.

Since satellite data can be considered big data there are some specific challenges involved. They can be mitigated by pre-processing these data into a data cube, which abstracts satellite images from a file-based format into a data structure that enables easy analysis. These so-called Earth observation data cubes are at the centre of this research. The aim was to contribute information about the resource usage of data cube generation because this aspect is largely unexplored at present.

Program code to generate data cubes using the two applications Euro Data Cube Batch Processing and mapchete Hub was written and the performance of the two was measured and compared. During the experiments, optical Sentinel-2 satellite imagery of one to 8.5-months was processed into a data cube with half-monthly time slices covering 17,000 km^2. The hypothesis was that the commercial service Batch Processing can outperform the custom mapchete Hub, which is based on open-source software and developed by EOX IT services, the sponsor of this thesis.

The experiments show that there is no clear winner in terms of size (mapchete Hub: 48.37 GiB; Batch Processing: 50.40 GiB) and storage costs (USD 1.185/USD 1.235) of the generated data cube. Also the processing time is similar up to the point where Batch Processing seems to hit a performance bottle neck and mapchete Hub is almost 50 % faster (1:24 vs. 2:03). Batch Processing's abstract currency *Processing Unit* is generally expensive to buy with real money and therefore mapchete Hub succeeds in the processing costs category by far as well. The only drawback of mapchete Hub is that there is no pre-defined UTM grid available, so that the user has to define a custom grid.

The advantage of Batch Processing's optimized UTM tiling grid especially shows in the extrapolation scenario for Austria, which spans two UTM zones. In this scenario EDC Batch Processing comes first in regard to processing time (14 % faster than mapchete Hub) and storage size/costs (17 % lower). Still mapchete Hub's processing costs are almost 200 times lower than Batch Processing's.

In conclusion, Batch Processing cannot outperform mapchete Hub in a business context due to its high processing costs. The results of the experiments allow for valuable insight into the performance of data cube generation and can be a reference for future work regarding tiling grids, optimizations for later data cube analysis and data cubes consisting of Sentinel-1 radar data.

Kurzfassung

Die Gemeinsame Agrarpolitik (GAP) hat relativ zu den anderen EU-Politiken einen der größten Budgetanteile. Das meiste Geld wird von den Zahlstellen der Mitgliedsländer in Form von Direktzahlungen an die Bauern ausgezahlt (2018: EUR 42 Milliarden). Im Gegenzug verpflichten sich die Bauern zu umweltfreundlicher Bewirtschaftungsweise und Landschaftspflege. (EU, 2020c) Die Erfüllung der Vorgaben wird mit zeitaufwändigen Vor-Ort-Kontrollen überwacht, die nur eine Kontrolle von 5 % aller Höfe erlaubt. (Devos, Fasbender, Griffiths, et al., 2017, p. 4)

Hier kommt das Copernicus-Programm mit seinen Sentinel-1- und -2-Satelliten ins Spiel. Um die GAP zu modernisieren und ihre Effizienz zu steigern, möchte die EU auf *Checks by Monitoring* (CbM) setzen und automatische Kontrollen aller Agrarflächen durchführen. Dies wird ermöglicht, indem ein Machine-Learning-Modell mit klassifizierten Satellitendaten trainiert wird. Das Modell klassifiziert dann wiederum Felder nach der darauf angebauten Kulturpflanze und vergleicht das Ergebnis mit den Angaben der Bauern. Nicht übereinstimmende Agrarflächen werden genauer kontrolliert.

Da Satellitendaten als *Big Data* erachtet werden können, gehen sie mit einigen spezifischen Herausforderungen einher. Diese können durch Vorprozessierung zu Datenwürfeln bewältigt werden. Satellitenbilder werden so von einem Datei-basierten Format in eine Datenstruktur abstrahiert, die eine mühelose Analyse ermöglicht. Die so genannten Erdbeobachtungsdatenwürfel bilden den Hauptfokus dieser Arbeit. Da der Ressourcenverbrauch der Generierung solcher Datenwürfel zurzeit fast unerforscht ist, war das Ziel, Informationen zu diesem Forschungsfeld beizutragen.

Dafür wurde Programmcode zur Datenwürfelgenerierung mittels der beiden Anwendungen *Euro Data Cube* (EDC) *Batch Processing* (BP) und *mapchete Hub* (mHub) geschrieben und deren Performance gemessen und verglichen. Die Experimente umfassten die Prozessierung von optischen Sentinel-2-Satellitenbildaufnahmen über einen Zeitraum von einem bis 8,5 Monaten und einer Ausdehnung von 17.000 km² in einen Datenwürfel mit halbmonatigen Zeitschichten. Die Hypothese war, dass der kommerzielle BP-Dienst eine bessere Performanz als die individuelle mHub-Lösung bietet, die auf quelloffener Software basiert und von dem Unternehmen EOX IT Services entwickelt wird, dem Sponsor dieser Masterarbeit.

Die Experimente zeigen, dass es bei der Speichergröße (mHub: 48,37 GiB/BP: 50,40 GiB) bzw. der -kosten (USD 1,185/USD 1,235) des generierten Datenwürfels keinen klaren Sieger gibt. Bis zu einem Performanz-Engpass von BP ist auch die benötigte Zeit ähnlich, danach ist mHub beinahe 50 % schneller (1:24/2:03). Die abstrakte Währung von BP ist im Allgemeinen teuer in Bezug auf reales Geld, wodurch mHub auch bei den Prozessierungskosten bei weitem voranliegt. Der einzige Nachteil von mHub ist das Fehlen eines vordefinierten UTM-Rasters, das daher selbst erstellt werden muss.

Der Vorsprung des optimierten UTM-Kachelrasters von *BP* zeigt sich vor allem im Extrapolationsszenario für ganz Österreich, welches in zwei UTM-Zonen liegt. In diesem Szenario gewinnt EDC BP in den Kategorien Prozessierungszeit (14% Vorsprung) und Speichergröße/-kosten (17%). Dennoch kostet die Prozessierung mit mHub fast 200 Mal weniger.

Letztendlich bietet BP in einem Geschäftsumfeld aufgrund seiner höheren Kosten keine höhere Performanz als mHub. Die Ergebnisse der Experimente erlauben wertvolle Einblicke in die Performanz der Datenwürfelgenerierung und können eine Grundlage für zukünftige Forschung bezüglich Kachelraster, Optimierungen für die Datenwürfelanalyse und Datenwürfel mit Sentinel-1-Rasterdaten sein.

Content

The source code that was produced in the context of this master's thesis can be found on Github: https://github.com/StefanBrand/masterdatacube

List of figures

List of tables

List of formulas

List of abbreviations

AaaS	Analytics as a Service
ADBMS	Array Database Management Systems
AMA	Agrarmarkt Austria
AMD	Advanced Micro Devices
AMS	Area Monitoring System
AOI	Area of Interest
API	Application Programming Interface
AWS	Amazon Web Services, Amazon Web Services
BNDVI	Blue Normalized Difference Vegetation Index
BoA	Bottom-of-Atmosphere
BP	Batch Processing
CAP	Common Agricultural Policy
CbM	Checks by Monitoring
CDM	Common Data Model
CEO	Chief Execution Officer
CF	Climate and Forecast
CLI	Command-Line Interface
COG	Cloud-Optimized GeoTIFF
CPU	Central Processing Unit, Central Processing Unit
CRS	coordinate reference system
CVI	Chlorophyll vegetation index
DaaS	Desktop as a Service
DGGS	Discrete Global Grid Systems
DIAS	Data and Information Access Service
DSaaS	Data Storage as a Service
EC2	Elastic Compute Cloud
EDC	Euro Data Cube, Euro Data Cube
EMS	Electromagnetic Spectrum
EODC	Earth Observation Data Cube
EPSG	European Petroleum Survey Group Geodesy
ESA	European Space Agency
EU	European Union
EUMETSAT	European Organisation for the Exploitation of Meteorological Satellites
GCP	Google Cloud Platform
GEE	Google Earth Engine
GFS	Google File System
GHz	Gigahertz
GiB	Gibibyte
GNDVI	Green Normalized Difference Vegetation Index
GPU	Graphics Processing Unit
GSAA	Geospatial Aid Application
GUI	Graphical User Interface
HB	Higher Better metrics
HDF	Hierarchical Data Format
HDFS	Hadoop Distributed File System
HPC	High-Performance Computing
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IACS	Integrated Administration and Control System
IAMaaS	Identity and Access Management as a Service
INSPIRE	Infrastructure for Spatial Information in the European Community
IT	Information Technology
JEODPP	JRC Earth Observation Data and Processing Platform
JPEG	Joint Photographic Experts Group
JRC	Joint Research Center, Joint Research Centre
JSON	JavaScript Object Notation
L1C	Level-1C
L2A	Level-2A
LB	Lower Better metrics
LPIS	Land Parcel Identification System
MaaS	Monitoring as a Service
MAB	Monitoring Algorithm Baseline, Monitoring Algorithm Baseline
MERIS	Medium Resolution Imaging Radiometer
mHub	mapchete Hub
ML	Machine Learning
MODIS	Moderate Resolution Imaging Spectroradiometer
NASA	National Aeronautics and Space Administration
NB	Nominal Better metrics
NDSI	Normalized Difference Salinity Index
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
netCDF	Network Common Data Form
NIR	Near-Infrared Region
NIST	National Institute of Technology
ODC	Open Data Cube
OGC	Open Geospatial Consortium
OLAP	OnLine Analytical Processing
OSD	Object-based Storage Devices
OTSC	On-The-Spot Checks
OWL	Web Ontology Language
PA	Paying Agency
PaaS	Platform as a Service
PiB	Pebibyte
PNG	Portable Network Graphics
PoP	Point of Presence
PU	Processing Unit
radar	radio detection and ranging

RDF	*Resource Description Framework*
REST	Representational State Transfer
RS	Remote Sensing
S1	Sentinel-1
S2	Sentinel-2
S2GM	Sentinel-2 Global Mosaic
S3	*Simple Storage Service*
SaaS	Software as a Service
SAR	synthetic aperture radar
SCL	*Scene Classification Layer*
SecaaS	Security as a Service
SECO	Software Ecosystem
SEPAL	System for Earth Observation Data Access, Processing and Analysis for Land Monitoring
SH	Sentinel Hub
SLAR	side-looking airborne radar
SoS	System of Systems
SOW	Statement of Work
SPOT	System Probatoire pour l’Observation de la Terre
SRS	Satellite Remote Sensing
SWIR	Shortwave Infrared Region
TiB	Tebibyte, Tebibyte
TIFF	Tagged Image File Format
TIR	Thermal Infrared Region
TM	Thematic Mapper
ToA	Top-of-Atmosphere
ToC	Top-of-Canopy
uint16	16-bit unsigned integer
USGS	United States Geological Survey
UTM	Universal Transverse Mercator
UV	Ultraviolet
vCPU	virtual CPU
VDI	Virtual Desktop Infrastructure
VIS	Visible
WCPS	Web Coverage Processing Service
WMTS	Web Map Tile Service
YAML	YAML Ain't Markup Language

1 Introduction

1.1 Motivation

Food security is a recurring topic in a nowadays world struck by climate change, but it has been a concern for nations for a long time. Almost from their beginning the European Communities have dedicated large parts of their budget to supporting farmers and the food supply under the so-called Common Agricultural Policy (CAP) that was founded in 1962. In 2018 about 36 % (EUR 58.82 billion) of the European Union’s (EU) budget were spent on the CAP, of which direct payments to farmers had the greatest share (EUR 41.74 billion), followed by the other two CAP areas rural development (EUR 14.37 billion) and market measures (EUR 2.7 billion). The direct payments benefit around 10 million farms with 22 million people working regularly in the sector; in return the farmers commit to environmentally friendly farming practices and preservation of the countryside. (EU, 2020c)

Payments to farmers are administered by the EU member states through one or multiple *paying agencies* (PAs) per country. To ensure standardized handling of subsidies across the European Union a system named Integrated Administration and Control System (IACS) has been implemented. IACS consists of databases and applications to track animal and farm land stock: Besides a database for animals, there is a database for all agricultural parcels in the EU (Land Parcel Identification System, LPIS) and a graphical tool that helps farmers declare their cultivated crops (geospatial aid application, GSAA). The control component of IACS comprises administrative checks of all applications through computerised cross checks and physical on-farm checks of a sample of farmers (on-the-spot checks, OTSC). (EU, 2020b)

The IACS process is conducted yearly; in Austria the deadline for the farmers’ declarations is the May 15th of each year (AMA, 2020, p. 11) and checks for validity are carried out until the end of the crop season on November 30th. The OTSC sample size usually is 5% of the farms (Devos, Fasbender, Griffiths, et al., 2017, p. 4), which means that on average an inspector only visits a farm every 20 years. First of all, this is accompanied by a lot of manual administrative work for the PAs and the farmers and, secondly, irregularities that have arisen on a farm—be it with bad intent or inadvertently—might not be discovered for years.

A new development, which can make IACS much more efficient and just, is made possible by the newly available Copernicus Programme, specifically through its Sentinel-1 (S1) and Sentinel-2 (S2) satellites. The satellite pair S1A (launched in 2014 being the first of almost 20 more Copernicus satellites until 2030) and S1B is equipped with radar sensors, which enable calculation of crop biomass and detection of crop harvesting. The S2 constellation similarly consists of two satellites with optical sensors that capture multispectral images of the Earth’s surface. This allows for crop type detection, crop health analysis and the monitoring of land use change. The combined frequent observations of S1 (at least two days revisit time) and S2 (3–4 days), together with geo-tagged photos, drone images and supplementary documentation by farmers (e.g. seed labels), facilitate automated IACS checks that are commonly referred to as *checks by monitoring* (CbM) by the European Commission’s Joint Research Centre (JRC) and EU legislation. (Devos, Fasbender, Griffiths, et al., 2017; Devos, Fasbender, Lemoine, et al., 2017; ESA, 2018a; EU, 2018a, 2018b, 2020a) Sometimes the JRC also labels this new scheme of checks *Area Monitoring System* (AMS). (Loudjani, 2019)

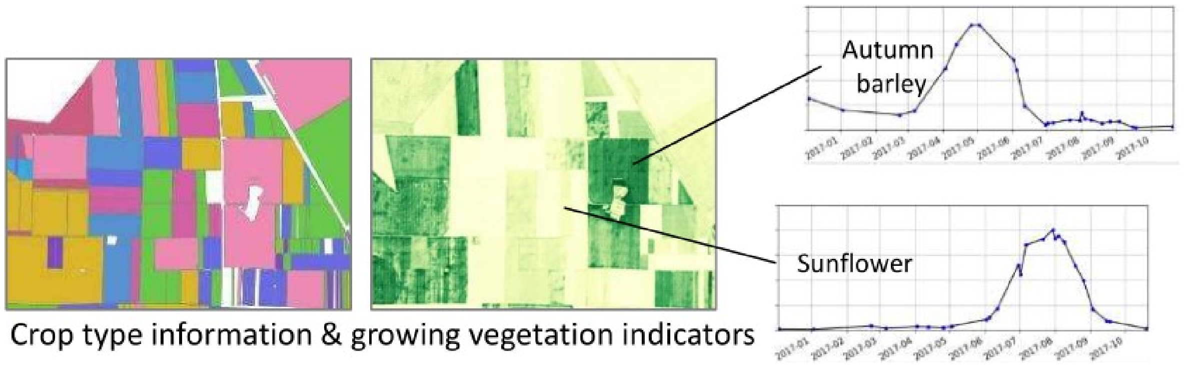


Fig. 1: Typical vegetation index time series of autumn barley and sunflower deduced from satellite observations. Source: Koetz et al., 2019, p. 20

In order to mine information from satellite imagery, machine learning (ML) algorithms are employed. In general, data scientists *train* an ML model and apply this model to new data. In terms of CbM, satellite imagery is combined with agricultural parcels from the LPIS that have crop type labels attached. The ML model then learns typical vegetation index time series for each crop type (see Fig. 1). When new satellite imagery is presented to the algorithm, it can—with certain confidence—determine the crop type of unlabeled parcels. These *predictions* are compared to the GSAA farmers' declarations and checked for conformity (see Fig. 2). If the GSAA and the predicted crop type do not match, an alarm is raised with the paying agency and further investigations are triggered.

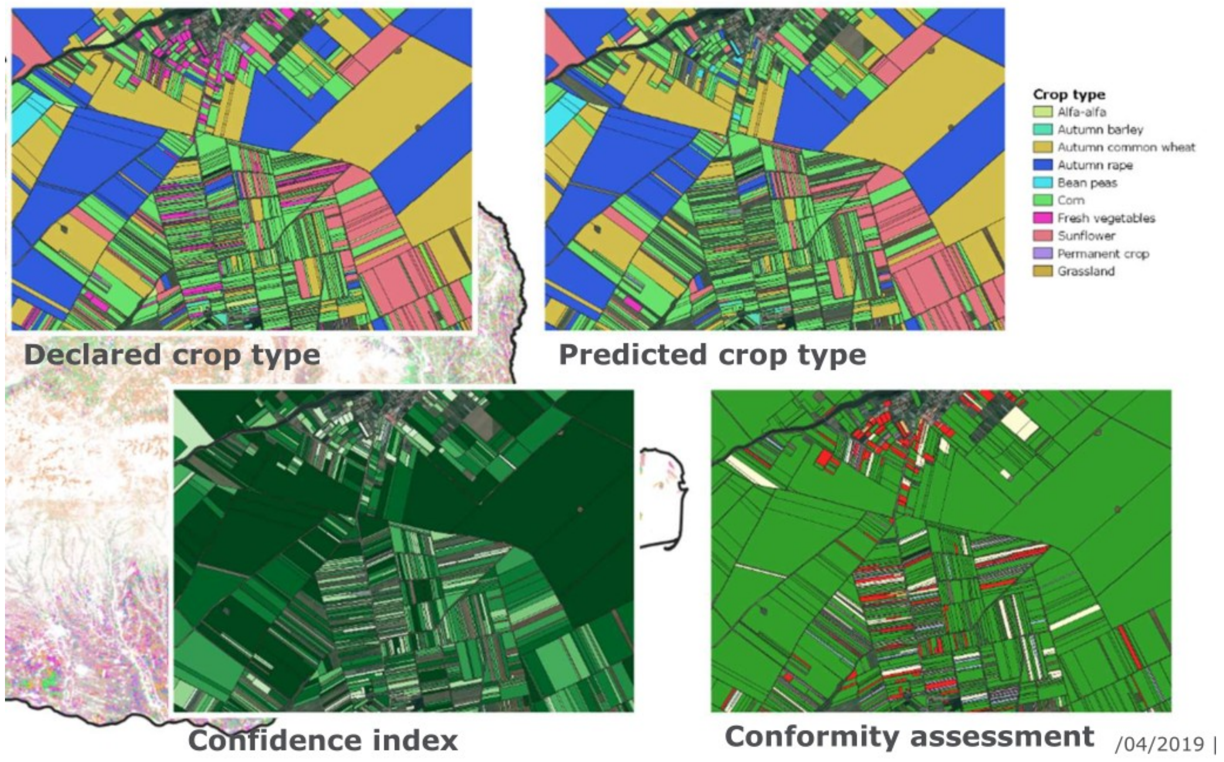


Fig. 2: Example maps for declared crop type, predicted crop type, confidence index and conformity assessment. Source: Koetz et al., 2019, p. 16

1.2 Problem

CbM can lead to huge efficiency gains for agriculture administrations, but leveraging satellite data also poses a challenge for the authorities because they are currently not well-equipped to handle this type of data. Satellite imagery is deemed *big data* or *big Earth data* by a lot of scientist, whose challenges are generally explained by the three (Woodcock et al., 2016, p. 13; Giuliani et al., 2017, p. 101,111, 2018, pp. 8659, 8661; Giuliani, Camara, et al., 2019, p. 1f; Wu et al., 2018, p. 693; Augustin et al., 2019, p. 1; Giuliani, Masó, et al., 2019, p. 2), or sometimes four (Lewis et al., 2017, p. 289f; Wang et al., 2019, p. 152) or five (Sudmanns et al., 2020, p. 833f) Vs:

- **Volume:** From beginning of the operations until the end of 2019 the Sentinels (1, 2, 3, 5P) archive contained almost 26 million products amounting to 17.23 PiB[1] of data (Knowelden & Castriotta, 2020, p. 27)
- **Velocity:** During November 2019, on average, 30,471 new products with a volume of 18.47 TiB[2] were published per day (Knowelden & Castriotta, 2020, p. 27f)
- **Variety:** There is an ever increasing variety of public and commercial satellite sensors, raw and atmospherically corrected imagery and derived data products that pose challenges in combining the different products for greater insight (e.g. Sentinel-1 and Sentinel-2 for CbM)
- **Veracity** (Lewis et al., 2017, p. 289f; Wang et al., 2019, p. 152): A lot of things can go wrong in the Earth observation pipeline. For example, for Sentinel-2 there is an anomaly database listing on-board irregularities that usually cannot be recovered and processing issues that potentially can be fixed after having been identified. As a result, some satellite acquisitions might contain errors and later be removed from the archive.
- **Value** (Sudmanns et al., 2020, p. 833f): The Copernicus data is available to use for free and thus does not generate revenue for the European society. However, this openly accessible wealth of data fosters business opportunities for companies in many industries.

Earth observation data cubes have set out to solve these challenges by abstracting individual satellite products into a uniform view on the data. Geographical and temporal boundaries of satellite imagery are dissolved and offered to users as a hypercube with at least three dimensions (x-coordinate, y-coordinate, time), potentially covering the whole Earth and all satellite observations since the launch of the service. Often there are more than three dimensions because sensors capture multiple wavelength bands (i.e. red, green, blue of the visible light) and satellite data might carry height information. This data representation enables easy analysis of time series and custom areas of interest and is perfectly fit for CbM, which deals with the development of agricultural parcels over time.

1.3 Research Question and Aims

EOX, the sponsoring company of this master's thesis, and the Austrian paying agency AMA (Agrarmarkt Austria) have initiated the *Monitoring Algorithm Baseline* (MAB) project to investigate needs of and solutions to the PA's CbM efforts. AMA's aim is to update the ML model on a monthly basis throughout the crop season, which requires timely additions to an underlying data cube. Two cloud services to process satellite data into multi-dimensional data cubes are at hand: *mapchete Hub* (mHub), which is developed in-house, and the *Euro Data Cube* (EDC), of which EOX is a consortium member. Since this cooperation should eventually lead to a commercial data processing offer, resource costs incurring to EOX are of utmost relevance in order to select the better option and draft a business plan.

Of special interest is the resource usage during data cube generation (execution time, know-how, processing costs) as well as of storing the resulting data cube (required storage space, cloud storage costs). This results in the following questions:

[1] PiB = Pebibyte = 2^{50} bytes
[2] TiB = Tebibyte = 2^{40} bytes

- Can the commercial cloud service EDC outperform the custom mHub application?
- How can a data cube be generated in each of the two tools?
- What does performance even mean?

1.4 State of the Art

There are not many empirical accounts of resource usage of processing Earth observation data. One publication could be found by Wang et al. (2019), who created 7-band satellite mosaics covering 1,208,000 km² from Landsat-TM (Thematic Mapper) imagery using the Chinese cloud services provider OpenStack. Wang et al. (2019) processed 28 Landsat scenes amounting to about 10 GB of total storage space in North-Eastern China. The mosaic was calculated on a virtual cluster with up to ten nodes, each equipped with 8 virtual CPUs[3] and 16 GB memory. Using only one node the calculation took 350 minutes, whereas three nodes managed the workload within 100 minutes. From five nodes (ca. 80 minutes runtime) no speedup could be noticed and ten nodes even led to an increase in runtime because of the parallelization overhead.

Let me naively translate these performance measurements to the whole area of Austria (83,879 km²) and to the data requirements of the MAB project, which includes a data cube of 16 bands at S2 resolution (10 m time 10 m) and 30 time slices. The Landsat-TM instrument has six bands at 30 m resolution and one band at 120 m resolution (16 x fewer pixels). (NASA, 2020) Therefore first the values for one 10-m-resolution band (9 x more pixels than 30 m resolution) will be determined and then the other factors will be accounted for. Formula 1 shows the resulting estimation.

Formula 1: Extrapolation of Wang et al.'s results to the extent of Austria and a Sentinel-2 data cube with 30 time slices and 16 bands.

$$f(x) = \frac{x}{6 + \frac{1}{16}} * 9 * 16\ bands * 30\ time\ slices * \frac{83{,}879\ km^2}{1{,}208{,}000\ km^2} = x * 49.5$$

This means that it would have taken Wang et al.’s (2019) experiment equipment 66 hours to process 495 GB of satellite data. It has to be noted that the additional processing time for calculating vegetation indices is not included in Wang et al.’s (2019) research and they also do not disclose data type and file format of the output mosaic.

1.5 Methodology and Structure of the Thesis

In the course of this thesis the performance of data cube generation on both mHub and EDC will be tested using custom processing scripts applied to a sub region of Austria. The thesis will solely focus on Sentinel-2 imagery and not treat Sentinel-1 data. Chapter 0 will introduce the reader to fundamentals of computer system performance measurements, remote sensing, cloud computing, Earth observation data cubes and data representation formats. It also includes a section on mHub and EDC, respectively. Chapter 0 goes into detail about the methodology employed (used performance metrics, test area, data cube schema and the code used to generate the data cubes). The results of the experiments will be presented in chapter 0 and put into context in chapter 0. A summary of the results of the thesis and conclusions can be found in chapter 0.

[3] CPU = Central Processing Unit

2 Theory

This chapter lays the ground for the whole thesis. Important concepts are explained and the latest knowledge in the relevant fields is summarized. First of all, computer systems performance is thoroughly defined. Detailed chapters about the fundamentals of remote sensing and the Sentinel satellites follow. Thereafter the relatively new cloud computing paradigm is described, including in-depth information about Amazon Web Services. The literature about Earth observation data cubes is recapitulated and the two services for data cube generation, Euro Data Cube and mapchete Hub, are introduced. A section on in-memory and on-disk data representation concludes the theory chapter.

2.1 Computer Systems Performance

Evaluating the performance of a computer system is important because performance ultimately translates to cost efficiency. (Obaidat & Boudriga, 2010, p. 1) E.g. is it cheaper to upgrade a computer system because of performance gains or does a minute performance increase not justify high installation costs?

The major goals of performance evaluation are the following, according to Obaidat & Boudriga (2010, p. 5f):

i. Compare alternative system designs (“find quantitatively the best configuration”)
ii. Procurement (“find the most cost-effective system for a specific application”)
iii. Capacity planning (“meet future demands in a cost-effective manner”, e.g. website load)
iv. System tuning (“find the set of parameter values that produce the best system performance”)
v. Performance debugging (find performance *bottlenecks*, i.e. the reason why a computer system does not meet performance expectations)
vi. Set expectations (in the planning phase of future computer systems)
vii. Recognize relative performance (contextualize new generations of computer systems)

Three methods that are used throughout the development process of a system can be discerned when conducting performance evaluation: (i) analytical modelling, (ii) simulation, (iii) measurement and testing. Analytical models are the least cost-intense evaluation method, but they also render the lowest accuracy in comparison to real measurements. However, they are the only relevant means to obtain performance values in the early design stage of new computer systems. Simulations form a middle course in the sense that they do not require a prototype or a finished system and still provide good accuracy given representative input data. The effort for creating a simulation model is considerable, though. (Obaidat & Boudriga, 2010, p. 7f)

Performance can be formalized by various measures. In computer systems the general interest is on how many times an event occurs (count), how long a process takes (time) and how large a specific parameter is (size). Performance metrics can be derived from one or multiple measures. For example, the performance of a service can be indicated by the following metrics: (Lilja, 2000, p. 9; Obaidat & Boudriga, 2010, p. 9)

- productivity (“rate at which the service is performed”)
- responsiveness (“time needed to perform the service”)
- usage metrics (consumed resources)
- availability (uptime)
- reliability (“probability that the system survives until some time *t*”)

Depending on their scale, these metrics can be categorized in *Higher better metrics* (HB – e.g. productivity), *Lower better metrics* (LB – e.g. responsiveness) and *Nominal better metrics* (NB – e.g. utilization of a system component; a utilization of 100 % is not desirable because it would constitute a bottleneck for the whole system).

Lilja (2000, p. 10ff) write about the “characteristics of a good performance metric”. According to them these are:

i. Linearity (performance value should be proportional to the actual performance)
ii. Reliability (if one system is better rated than the other, it should always outperform the other system in real conditions)
iii. Repeatability (each execution of a performance evaluation should yield the same results under given conditions)
iv. Easiness of measurement (the metric should not be hard to measure because otherwise it is also prone to incorrect measurements)
v. Consistency (for comparability the definition of the metric should be the same across systems)
vi. Independence (the metric should not be susceptible of manipulations by system manufacturers)

Of these six Obaidat & Boudriga (2010, p. 8) only really mention *reliability* in their opinion on “a good performance metric”. While their other remark “it should be relevant or meaningful” could be anything, they offer a specific perspective from the point of view of performance modelling. In the first place “it should be possible to develop models” and then “the model [...] should not be difficult to estimate”. By the latter they add the model perspective to Lilja’s notion of *easiness of measurement*.

Examples of performance metrics include clock rate, MIPS, MFLOPS and SPEC, which Lilja (2000, pp. 12–15) discards as not meeting the characteristics of a good performance metric. *Execution time*, on the other hand, “satisfies all of the characteristics of a good performance metric”. However, both *CPU time* (“total time the processor actually spends executing the program”) and *wall clock time* (“total time the user would have to wait to obtain the results”) should be reported to allow users to gain an insight on the time a program spends waiting on other programs. In order to compare two computer systems *speedup* factor and *relative change* in percent can be calculated. (Lilja, 2000, pp. 17–21)

2.2 Remote Sensing

Remote sensing (RS) is defined as “discerning information about the Earth’s surface from afar without direct physical contact”. (Eamus et al., 2016, p. 155) This is possible because we can use sensors to measure electromagnetic radiation and analyse the resulting data products. The framework of remote sensing consists of the following components (Eamus et al., 2016, p. 157f):

i. A source of electromagnetic radiation (passive RS: sun or Earth itself; active RS: Radar and Lidar)
ii. The Earth’s surface (reflects, absorbs and emits radiant energy)
iii. The sensor instruments (measure optical, thermal and microwave signals “over a range of spatial, temporal and spectral resolutions”)
iv. Receiving stations (process and calibrate raw sensor signals)
v. The user community (scientists and companies)

2.2.1 Physical Foundations of Remote Sensing

In an attempt to understand remote sensing, the first distinction has to be made between *passive* and *active* RS. In passive RS a sensor measures (a) solar radiation that is reflected from the Earth's surface or (b) terrestrial energy that is emitted from the Earth (e.g. thermal and microwave radiation). Active RS is conducted when the sensor sends electromagnetic waves to Earth, which are then reflected on the Earth's surface. The returned signal is measured by the same sensor that has generated the signal. (Eamus et al., 2016, p. 167f) Fig. 3 illustrates active and passive RS.

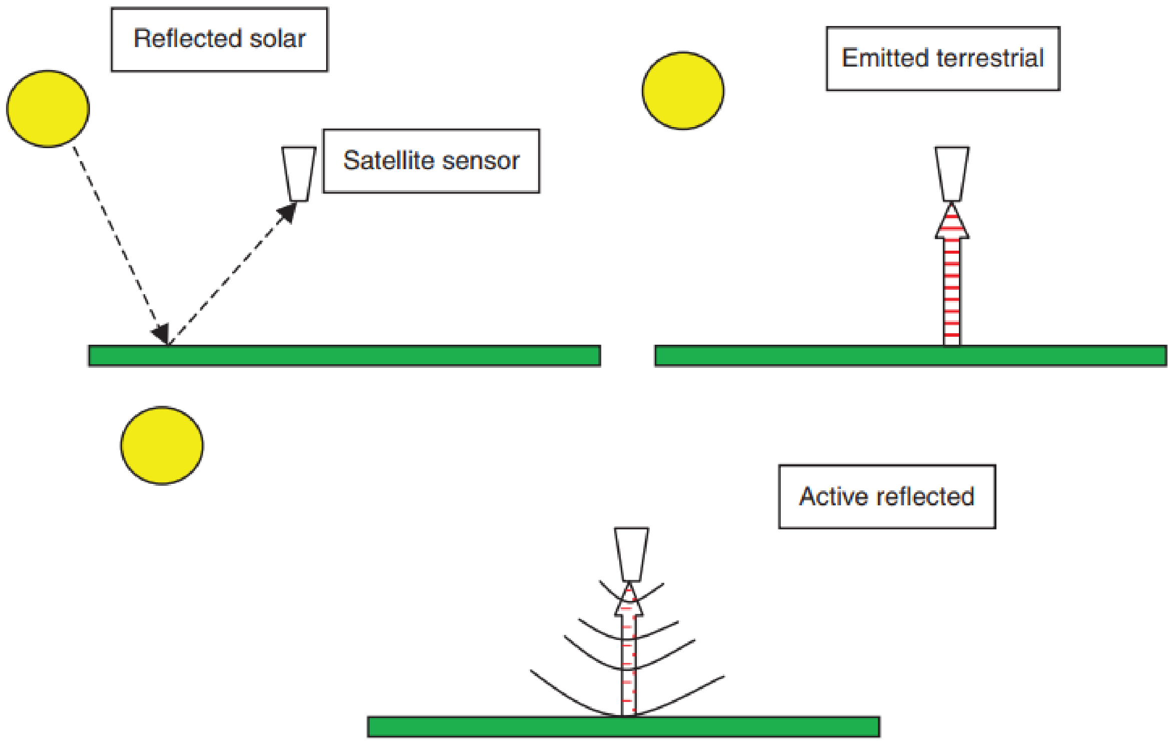


Fig. 3: Passive (above) and active (below) remote sensing. Source: Eamus et al., 2016, Figure 5.1

Optical, thermal and microwave signals have been mentioned before. Together they are part of the electromagnetic spectrum (EMS), which arranges electromagnetic waves according to their wavelengths (the distance between two peaks of the electric field) and—for microwave radiation—frequencies (speed of light divided by wavelength). The optical spectrum is further divided into the ultraviolet, visible and infrared regions (Eamus et al., 2016, p. 169ff; Moreira et al., 2013, p. 7):

i. UV: Ultraviolet region (0.1–0.4 µm →useful for atmosphere studies)
ii. VIS:Visible region (0.4–0.7 µm; visible to human eye → leaf pigments, surface water quality, soil minerals)
    a. blue (0.4–0.5 µm)
    b. green (0.5–0.6 µm)
    c. red (0.6–0.7 µm)
iii. NIR: Near-infrared region (0.7–1.3 µm → leaf structure and morphology)
iv. SWIR: Shortwave infrared region (1.3–8 µm)
    a. reflected solar radiation (1.3–3 µm → moisture content of vegetation and the upper soil surface)
    b. surface emitted signal (3–8 µm → high temperature sources, e.g. fires)
v. TIR: Thermal infrared region (8–14 µm → surface temperatures, vegetation stress, soil moisture, clouds, minerals, environmental contamination)

vi. Microwave region (>0.1 cm/<300 GHz) → penetrates clouds, forest canopies; "useful in analyses of soil surface moisture and roughness, as well as plant canopy moisture and roughness")
   a. X-band (2.5–4 cm/12–7.5 GHz)
   b. C-band (4–8 cm/7.5–3.75 GHz)
   c. L-band (15–30 cm/2–1 GHz)
   d. P-band (60–120 cm/0.5–0.25 GHz)

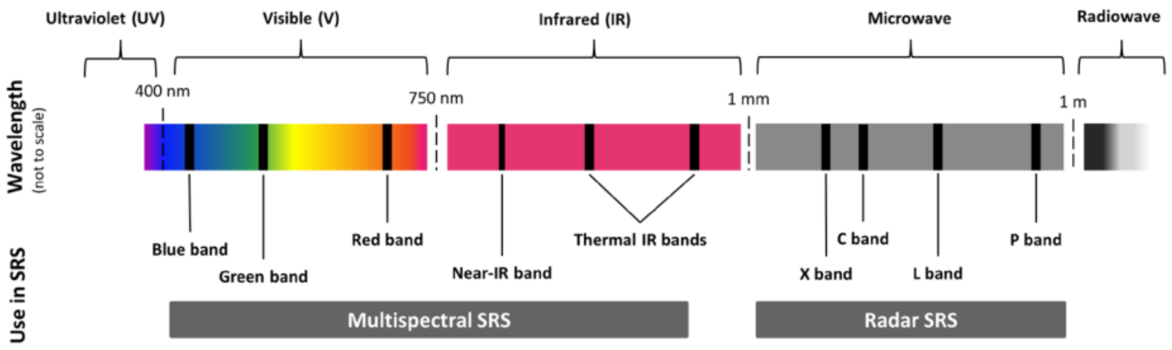


Fig. 4: The spectrum of electromagnetic radiation (not to scale), and its use in satellite remote sensing (SRS). Source: Pettorelli et al., 2018, Figure 2.2, cropped

As can be seen in Fig. 4, VIS and infrared regions can further be classified as *multispectral* satellite remote sensing (SRS) and the microwave region can be called *radar* SRS.

Energy can be reflected by, absorbed by or transmitted through an object and the reflected, absorbed and transmitted parts make up the total energy impacting on an object (*irradiance*). (Eamus et al., 2016, p. 172f) Just like any object, the atmosphere also absorbs, transmits (*atmospheric windows*) and reflects (*atmospheric scattering*) radiance. Atmospheric absorption is mainly related to the four gases *diatomic oxygen* (O_2; absorbs UV below 0.1 µm, small portions in TIR), *ozone* (O_3; absorbs UV below 0.3 µm, microwave at around 27 µm), *water vapour* (H_2O; absorbs SWIR at 1.45 µm, 1.95 µm, 6 µm, and small portion in NIR) and *carbon dioxide* (CO_2; absorbs TIR at 15 µm and SWIR between 2.5 and 4.5 µm). (Eamus et al., 2016, p. 180f)

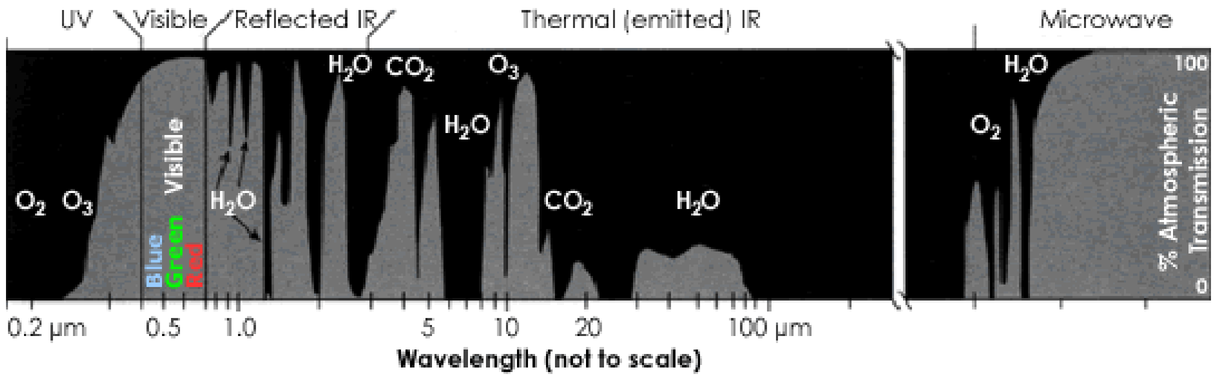


Fig. 5: Gases that affect atmospheric transmission. Source: King & Herring, 2001, cropped

The portions of the spectrum that are mostly transmitted by the atmosphere are called *atmospheric windows*. These are found in the VIS and NIR (0.3–1.35 µm) and in parts of the SWIR (1.5–1.8 µm; 2–2.4 µm; 2.9–4.2 µm; 4.5–5.5 µm), the TIR (8–14 µm) and the microwave region (>20mm). (Eamus et al., 2016, p. 181) Atmospheric absorption and transmission are illustrated in Fig. 5.

In contrast to absorption and transmission, which occur more often in some parts of the EMS than in others, *atmospheric scattering* influences sensor measurements in the whole EMS. It is caused by *aerosols*, small solid (smoke, smog, dust) and/or liquid (haze, fog) particles that are suspended in the atmosphere. Solar radiation that impinges on aerosols is scattered in all directions: One part diffusely reaches the Earth's surface and one part is reflected and "augments the signal received at the sensor." (Eamus et al., 2016, p. 182f) As a result, *atmospheric correction* has to be applied to turn *top-of-atmosphere* (ToA) reflectance into *bottom-of-atmosphere* (BoA) reflectance.

2.2.2 Landscape Biophysical Properties and their Characterization

Vegetation, soil, water and other objects on the Earth's surface have characteristic absorption, transmission and reflectance properties along the EMS. For example, the *spectral signature* of water shows that it does not reflect waves in the NIR, while vegetation particularly well reflects NIR waves (Fig. 6). (Eamus et al., 2016, p. 206) This discrepancy between the reflectance in different wavebands is used to calculate vegetation and vegetation water indices. (Eamus et al., 2016, pp. 217ff, 225) Among the most commonly used indices are the Normalized Difference Vegetation Index[4] (NDVI) and the Normalized Difference Water Index[5] (NDWI). Numerous other indices exist; the Index DataBase counts 519 different indices. (IDB, 2020) Eamus et al. (2016, p. 225) conclude their section on vegetation and vegetation water indices by stating that "[t]he use of multiple [vegetation indices] offers a more complete characterization of canopy properties."

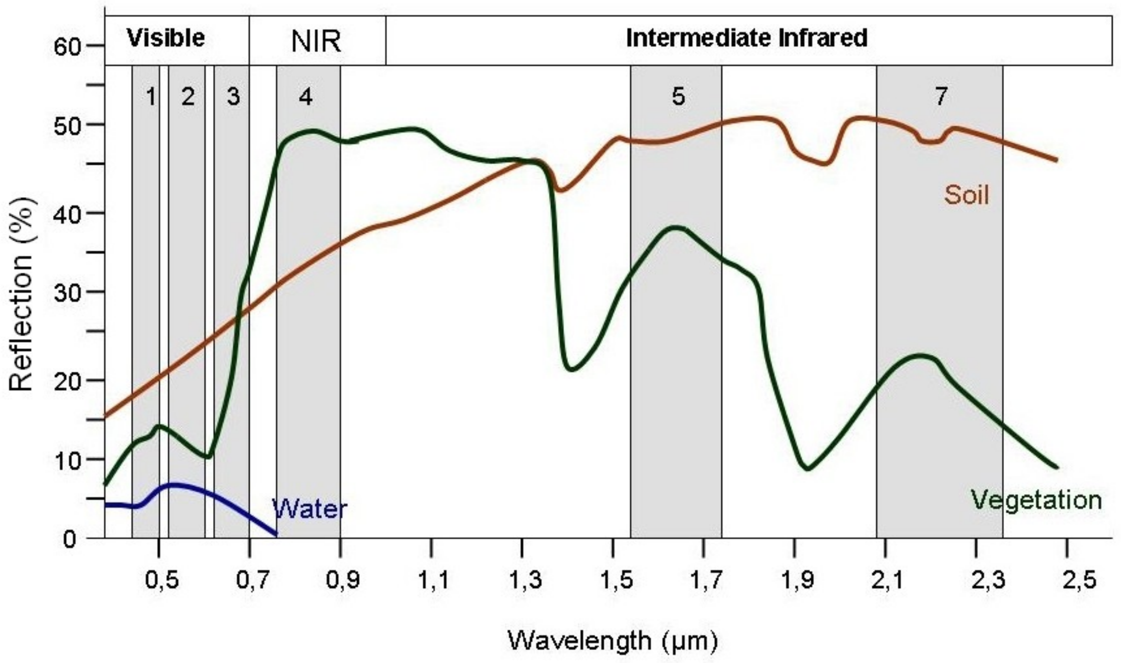


Fig. 6: Spectral signatures of soil, vegetation and water, and spectral bands (see below) of LANDSAT 7. Source: Siegmund & Menz, 2005 (as cited in SEOS, n.d.)

2.2.3 Sensors

In SRS, sensors mounted on satellites are used to measure electromagnetic radiance reflected on or transmitted by the Earth's surface. Each sensor has a set of resolution properties and optimizing one

[4] $(\rho_{NIR} - \rho_{red})/(\rho_{NIR} + \rho_{red})$

[5] $(\rho_{GREEN} - \rho_{NIR})/(\rho_{GREEN} + \rho_{NIR})$ (McFeeters, 1996) (as cited in (Du et al., 2016, p. 5))

sensor resolution leads to trade-offs in the remaining sensor properties (especially spatial vs. temporal resolution) (Eamus et al., 2016, pp. 185–189):

i. Spatial resolution: captured surface area per pixel, e.g.:
    a. Fine (0.5–5 m length of pixel side)
    b. Moderate (5–100 m)
    c. Coarse (0.1–100 km)
ii. Spectral resolution: “number of wavebands, their bandwidths, and overall spectral coverage”
    a. Multispectral (4–36 bands)
    b. Hyperspectral (up to 220 bands)
iii. Temporal resolution: “frequency of sensor observation over a given area on Earth” (from 10–15 minutes for geostationary meteorological satellites up to a month for fine spatial resolution sensors)
iv. Radiometric resolution: number of radiance levels captured by sensor, e.g.:
    a. 8 bit = 2^8 = 256 discriminable values
    b. 10 bit = 2^{10} = 1,024
    c. 16 bit = 2^{16} = 65,536

Sensor systems can further be classified by the orbital properties of the satellite. *Geostationary* satellites orbit the earth at a distance of 36,000–41,000 km and always stay fixed at one position above the Earth’s surface, moving as fast as the Earth rotates. One example for geostationary satellites is the Meteosat series of satellites for weather observation. The other important class of satellite orbits is comprised of *polar orbiting* satellites. These orbit the Earth “close to the poles, at altitudes of 600–950 km.” Notable examples for polar orbiting satellites include the SPOT[6]-VEGETATION, MERIS[7], MODIS[8], Landsat, Ikonos and QuickBird platforms. (Eamus et al., 2016, pp. 191–198)

A special type of sensors is active microwave sensors, also called *radar* (radio detection and ranging). Imaging radars can further be divided into (a) real aperture radar or side-looking airborne radar (SLAR) and (b) synthetic aperture radar (SAR). Pulses of microwaves are generated and sent to Earth and the *backscattered* signal’s amplitude and phase are subsequently measured by the radar sensor. The backscattered signal is mainly influenced by the electrical and physical (geometry, roughness) properties of the Earth’s surface, which allows drawing conclusions about roughness and moisture content of soil and vegetation and about topography. (Eamus et al., 2016, p. 199; Moreira et al., 2013, p. 7) An important technique is *interferometry*, whose idea “is to compare for a given scene the phase of two or more complex radar images that have been acquired from slightly different positions or at different times.” The deferred metric called *coherence* “describes the degree of correlation between [...] two radar images” and can be used to analyse land cover and land use change. (Moreira et al., 2013, p. 19ff)

2.2.4 Level of Processing

Different processing steps are applied to the data that is transferred from satellite sensors. The resulting data products are classified into several *levels of processing*. Chuvieco (2020, p. 201) describes 5 levels of processing ranging from level 0 to level 4. Their characterization is reproduced in Table 1.

[6] System Probatoire pour l’Observation de la Terre
[7] Medium Resolution Imaging Radiometer
[8] Moderate Resolution Imaging Spectroradiometer

Level	Characterization
0	Raw data without any corrections, “may include different errors and artifacts”
1	Data with geometric and radiometric corrections, e.g. sensor calibration (converting measured digital numbers into actual radiance values), applied. May include “generation of top of the atmosphere (ToA) temperature or reflectance”.
2	Data of atmospherically corrected top-of-canopy (ToC)/bottom-of-atmosphere (BoA) reflectance (multispectral) and temperature (TIR)
3	Data resampled to a certain standard grid (spatially and/or temporally aggregated)
4	Data of derived variable (evapotranspiration, land cover, ...)

Table 1: Levels of processing of satellite data. Source: Chuvieco, 2020, p. 201

2.3 Sentinel Satellites

The *Sentinels* are a set of Earth observation satellites that form part of the EU’s Copernicus programme. Each of the six missions is made up by a pair of satellites, doubling temporal resolution of each platform, respectively. Sentinel-5P is an exception in this regard. It was launched as a precursor for Sentinel-5 to bridge the gap between the Envisat satellite and the Sentinel-5 mission. (ESA, 2020c; European Commission, n.d.) Table 2 shows an overview of the Sentinel missions.

Name		Dedicated/ onboard	Operated by	Usage scenario	Launch years
Sentinel-1		Dedicated	ESA[9]	Radar imaging	2014/2016
Sentinel-2		Dedicated	ESA	Multispectral imaging for land monitoring	2015/2017
Sentinel-3		Dedicated	EUMETSAT[10]	Sea-surface topography, sea- and land-surface temperature, ocean/land colour	2016/2018
Sentinel-4		Onboard	EUMETSAT	Atmospheric monitoring	N/A
Sentinel	-5P	Dedicated	ESA	Trace gases and aerosols for air quality/climate monitoring	2017
	-5	Onboard	EUMETSAT	Atmospheric monitoring	N/A
Sentinel-6		Dedicated	EUMETSAT	Radar altimeter for global sea-surface height	N/A

Table 2: Overview of the Sentinel missions. Dedicated/onboard: Whether the instrument has its own platform or is on board some other satellite platform. Sources: ESA, 2020c; European Commission, n.d.

While both Sentinel-1 and Sentinel-2 products should be used for accurate results in the context of checks by monitoring, this thesis focuses on the Sentinel-2 mission and data. The Sentinel-2 mission is composed of two satellites (Sentinel-2A/B) and its multispectral instrument features a spectral resolution of 13 bands (four 10-m bands, six 20-m bands and three 60-m bands). (Chuvieco, 2020, p. 90; ESA, 2020e) An overview of the bands is presented in Table 3.

[9] European Space Agency
[10] European Organisation for the Exploitation of Meteorological Satellites

Band number	Approx. central wavelength (nm)	Spatial resolution	Application
1	443	60	Aerosol detection (atmospheric correction)
2	493	10	Blue
3	560		Green
4	665		Red
5	704	20	Vegetation red edge
6	740		
7	783		
8	833	10	NIR
8a	865	20	Vegetation red edge
9	945	60	Water vapour (atmospheric correction)
10	1374		Cirrus detection
11	1610	20	SWIR – snow/ice/cloud detection, vegetation moisture stress assessment
12	2190		

Table 3: Overview of Sentinel-2 bands with their spectral and spatial characteristics and applications. Source: Chuvieco, 2020, p. 90; ESA, 2020e

As illustrated in Fig. 7 wavebands are not always equally spaced in the electromagnetic spectrum. For example, the narrow 8a-band captures radiation extremes that would be flattened out by the wide NIR-band no. 8.

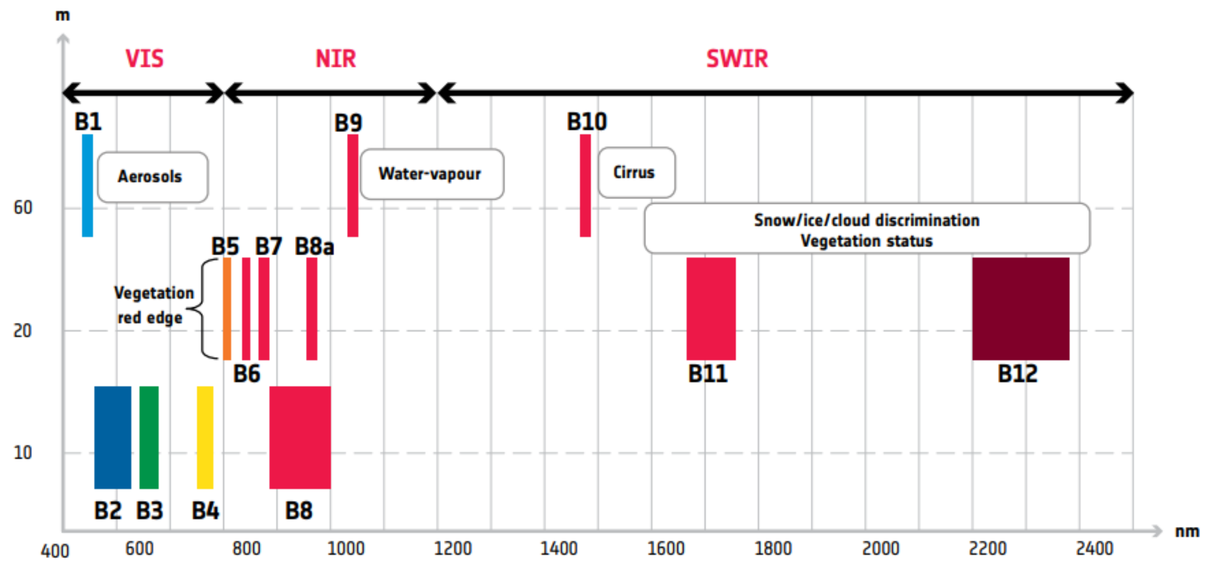


Fig. 7: Sentinel-2: Spatial resolution vs. wavelength. Source: ESA, 2015a, p. 8

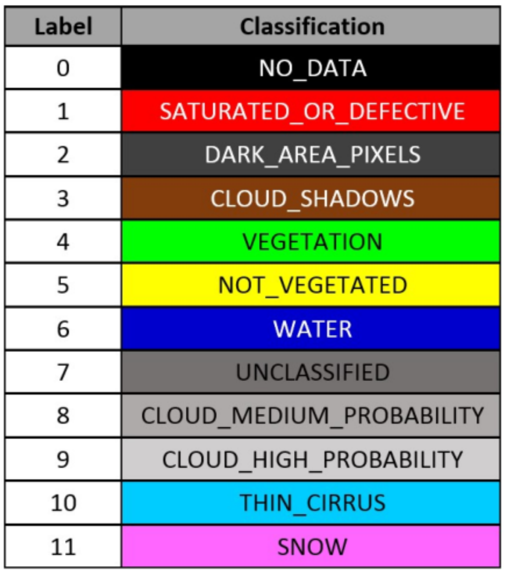
Concerning the temporal resolution the pair of Sentinel-2 satellites has a combined revisit frequency of at least 5 days (10 days in Antarctica and non-continental Arctic regions). Since satellite swaths overlap in higher latitudes, the revisit frequency in Austria, for example, can go up to three observa-tions per week. (ESA, 2020f)

When flying over a continent, the satellites continuously acquire data; this is called a “datatake” The datatakes are processed by a ground segment and distributed for free as 100 x 100 km “granules” in the UTM[11]/WGS84 projection. Two versions differing in their level of processing are made available

[11] UTM = Universal Transverse Mercator

to the public: Level-1C (L1C; ToA, about 600 MB each) and Level-2A (L2A; BoA, about 800 MB each). (ESA, 2020d)

In the process of calculating the L2A product, not only atmospheric correction but also a scene classification is performed. After resampling to a 60-m spatial resolution, pixels are divided up into cloud-free, cloudy (with multiple probability levels) and shadowy pixels. The cloud-free pixels are further classified as being vegetated, bare, water or covered by snow. The legend of the resulting classification map in Fig. 8 also shows an entry for saturated or defective pixels. These are excluded from the processing steps from the beginning.



Label	Classification
0	NO_DATA
1	SATURATED_OR_DEFECTIVE
2	DARK_AREA_PIXELS
3	CLOUD_SHADOWS
4	VEGETATION
5	NOT_VEGETATED
6	WATER
7	UNCLASSIFIED
8	CLOUD_MEDIUM_PROBABILITY
9	CLOUD_HIGH_PROBABILITY
10	THIN_CIRRUS
11	SNOW

Fig. 8: S2A Scene Classification Values. Source: (ESA, 2020b)

The official outlet for L1C and L2A Sentinel-2 data is the Copernicus Open Access Hub, but Sentinel data can also be downloaded from one of the four Data and Information Access Services (DIAS) or from an S3[12] storage in the AWS (Amazon Web Services) cloud. (AWS, 2020q; ESA, 2020a)

2.4 Cloud Computing

Outsourcing computation into the "cloud" is a new[13] paradigm that follows the *distributed computing* paradigm (e.g. computing on multiple computers that are connected via a network). (Murugesan & Bojanova, 2016, p. 4) The US National Institute of Standards and Technology (NIST) defines *cloud computing* as

> "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell & Grance, 2011, p. 2)

This definition translates to some key characteristics of computing clouds: (Mell & Grance, 2011, p. 2; Murugesan & Bojanova, 2016, p. 5)

[12] S3 stands for „Simple Storage Service" (AWS, 2020g)
[13] Amazon "pioneered the cloud IaaS market in 2006". (Bala et al., 2019)

- **On-demand self-service:** Unilateral provisioning of processing and storage resources
- **Broad network access:** Cloud capabilities can be accessed from any device (e.g. smartphone, tablet, laptop, workstation)
- **Resource pooling:** Physical and virtual resources are dynamically assigned and reassigned to multiple users depending on demand
- **Multitenancy:** Multiple users (tenants) use the same physical resources
- **Rapid elasticity and scalability:** Seemingly infinite resources are scaled up and down rapidly commensurate with demand
- **Measured service:** Built-in monitoring and reporting and pay-per-use pricing schemes

Virtualization in this regard is a concept that allocates physical infrastructure to multiple users enabling more efficient resource utilization. For example, an eight-core CPU (central processing unit) could be offered to two customers, who each need a four-core CPU.

Computing clouds can be categorized into three foundational service models that are offered and used individually or in combination with each other and additional services. The three service models are described here (ordered from less control to more control): (Mell & Grance, 2011, p. 2f; Murugesan & Bojanova, 2016, p. 6f)

- **Software as a Service (SaaS):** Web apps; users do not have to install local software, but only have limited configuration options for the app's behaviour and cannot control the underlying cloud infrastructure.
- **Platform as a Service (PaaS**): Users can deploy their own applications in the cloud environment using the supported programming languages and libraries, but have no control of the operating system and the cloud infrastructure.
- **Infrastructure as a Service (IaaS**): Raw computer infrastructure (servers, CPU, memory, storage, etc.) is provisioned to the users depending on their requirements and they can install and run arbitrary software (operating systems and applications). Instead of buying expensive hardware, customers are billed on a per-use basis.

There are several cloud support services that complement the three service models to a full cloud ecosystem, for example: (Murugesan & Bojanova, 2016, p. 7ff)

- **Data Storage as a service (DSaaS**): Data is stored in a virtual storage location and is backed up and secured by the provider to avoid data loss and theft, respectively
- **Analytics as a Service (AaaS**): Software and tools for analysis and mining of big data
- **Desktop as a Service (DaaS**): Provision of a virtual desktop infrastructure (VDI); desktop environments are streamed via the internet
- **Security as a Service (SecaaS):** Providers either secure cloud infrastructure or the customers' on-premises systems using, for example, virus detection, intrusion detection and encryption
- **Identity and Access Management as a Service (IAMaaS):** Cloud-based "user provisioning, authentication, authorization, self-service, password management, and deprovisioning"
- **Monitoring as a Service (MaaS**): Continuous state monitoring of online services

Further, cloud systems can be classified according to their deployment model (ordered from less private to more private): (Murugesan & Bojanova, 2016, p. 9)

- **Public cloud:** The most popular cloud deployment model; open to anyone
- **Community cloud:** Optimized for a particular industry sector
- **Virtual private cloud:** Virtual private segment of a public cloud with more control over the cloud infrastructure
- **Private cloud:** On-premises cloud for internal use; full control over applications and data

If an enterprise uses more than one of the above mentioned deployment models, the notion of *hybrid clouds* is being used. An enterprise might selectively use some public cloud services and store more sensitive data in a private cloud behind its own firewall. (Murugesan & Bojanova, 2016, p. 9f)

According to market research firm Gartner the leading players in the public cloud IaaS market as of July 2019 are Amazon with the Amazon Web Services (AWS), Microsoft with Azure and Google with the Google Cloud Platform (GCP) followed by the niche players Oracle, Alibaba Cloud and IBM. All of these six global vendors also include PaaS solutions in their portfolio. (Bala et al., 2019) On the contrary there are thousands of SaaS providers "from Adobe to Anaplan to Atlassian to Google to Microsoft to Okta to Oracle to Salesforce to SAP to Slack." (Knorr, 2020)

In the context of this master's thesis the most relevant cloud computing services are two IaaS offers of AWS: Elastic Compute Cloud (EC2) and Simple Storage Service (S3). These and other AWS services are located in 24 *regions* around the globe (3 more planned), which each have two to six *availability zones* amounting to a total of 77 (+9). These availability zones are physically separated from each other and are comprised of one or more data centre facilities. Customers can choose to deploy applications across multiple availability zones in the same region in order to increase service reliability. Content is served to end users via a network of 216 *Points of Presence* (PoP), which also exist in places that do not have AWS regions, e.g. in Eastern Europe. (AWS, 2020n, 2020o, 2020j) See Fig. 9 for an overview of the AWS infrastructure.

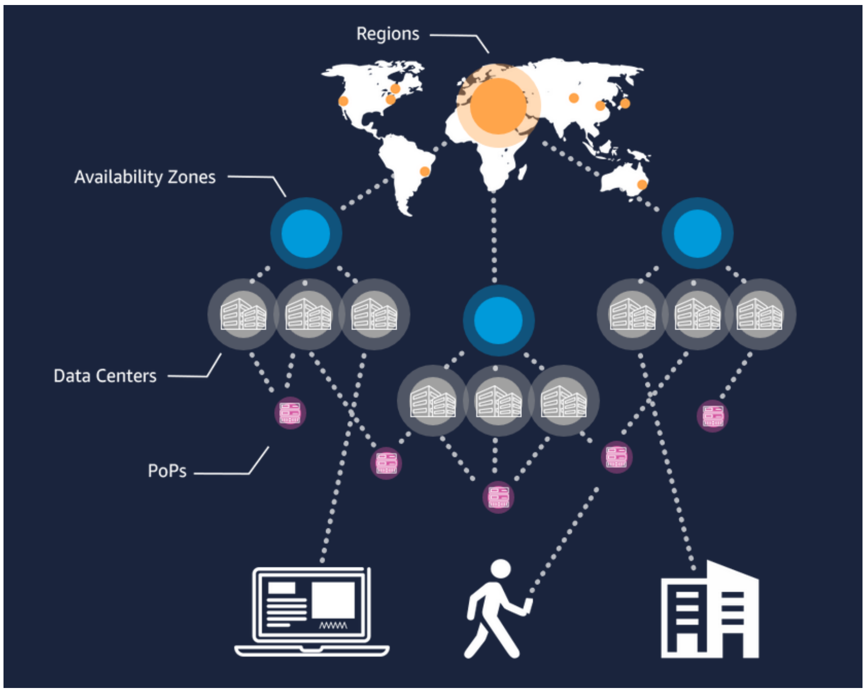


Fig. 9: AWS infrastructure diagram. Source: AWS, 2020j

EC2 is a virtual computing environment built around so-called *instances*. There is a range of different instance families, some for general purpose computing, some optimized for computing, memory or storage and some with GPU[14] support for “accelerated computing”. Each of the instance families is further differentiated by the number of vCPUs (virtual CPUs) and by the memory and storage available to the machines. Instance types might have names like *a1.large* (1 vCPU, 2 GiB[15] memory), *m5.4xlarge* (16 vCPU, 64 GiB) or *x1e.32xlarge* (128 vCPU, 3 904 GiB); altogether there are hundreds of distinct instance type configurations available. (AWS, 2020b, 2020c)

There are five billing options for the EC2 service: (AWS, 2020e)

- **On-demand:** Basic plan that is billed per second (Linux operating systems) or per hour (Windows). Example for *m5ad.4xlarge* with Linux in the Frankfurt region: USD 1.00 per hour. (AWS, 2020l)
- **Reserved instances:** Commitment to continued use of an instance with or without upfront and monthly payments for a 1- or 3-year period. Example: Between USD 0.588 and USD 0.630 effectively per hour for a 1-year period (37–41% less than on-demand price). (AWS, 2020m)
- **Spot instances:** Spare instances that can be reclaimed by AWS at any time when another customer has on-demand computing needs. Example: USD 0.2785 per hour (72% less). (AWS, 2020f)
- **Savings plans:** Similar to reserved instances, but instead of committing to a certain instance, customers commit to a monthly amount of money for a 1- or 3-year term and can flexibly distribute that amount among instances. Example: Between USD 0.762 and USD 0.817 for 1-year period (18–24% less). (AWS, 2020p)
- **Dedicated hosts:** A whole physical server is provisioned exclusively to one customer, which enables the customer to use existing server-bound software licences. On-demand, reservation and savings plans exist. (AWS, 2020a)

AWS S3 provides object-based cloud storage to customers. An *object storage* stores data as objects instead of as blocks. The equally-sized block structure features high access performance, but file system abstraction of conventional storage leads to a considerable overhead. The reason is that conventional file systems have to map the logical structure of files to the individual blocks in the storage medium. The variable size of objects solves this on a lower technical level by offloading the storage management from the file system to the storage device, combining the good performance of blocks with the highly granular security policies and easy manageability of files. Through this solution object-based storage devices (OSD) gain self-management, which “includes actions such as reorganizing data to improve performance, scheduling regular backups, and recovering from failures.” (Mesnier et al., 2003, pp. 84–90)

In S3 an object storage is called *bucket*. There are six storage classes with different properties and prices: (AWS, 2020i, 2020h, 2020k)

- **Standard:** General-purpose class with replication across multiple availability zones. Storage price for the first 50 TiB[16] in the Frankfurt region: USD 0.0245/GiB

[14] GPU = Graphics Processing Unit
[15] GiB = Gibibyte (2^{30} bytes)
[16] TiB = Tebibyte (2^{40} bytes

- **Standard-IA:** Infrequent Access (IA) storage class with lower storage fees (USD 0.0135/GiB), but higher prices for PUT, COPY, POST, LIST, GET and SELECT requests; additional data retrieval fee
- **One Zone-IA:** Same as Standard-IA except for missing replication. Storage price: USD 0.0108/GiB
- **Intelligent-Tiering:** Combination of frequent and infrequent access storage classes; objects are automatically moved to IA if they have not been accessed for 30 consecutive days. Price like Standard and Standard-IA with additional monthly Monitoring and Automation fee per 1 000 objects.
- **Glacier:** Low-cost storage option for long-term archiving with three differently-priced retrieval options (Expedited: 1–5 minutes retrieval time, Standard: 3–5 hours, Bulk: 5–12 hours). Storage fees: USD 0.0045/GiB
- **Glacier Deep Archive**: Same as Glacier, but with different retrieval times (Standard: 12 hours, Bulk: 48 hours). Storage fees: USD 0.0018/GiB

Besides, storage fees costs arise for *PUT*, *COPY*, *POST* and *LIST* requests, ranging from USD 0.0054 (Standard, Intelligent-Tiering) to USD 0.06 (Glacier, Glacier Deep Archive) per 1 000 requests, and *GET* and *SELECT* requests, ranging from USD 0.00043 (all but IA) to USD 0.001 (IA) per 1 000 requests. The IA and Glacier storage classes have additional fees for data retrieval. (AWS, 2020h)

2.5 Earth Observation Data Cubes

The concept of data cubes arose in the context of business intelligence and *online analytical processing* (OLAP) in the 1990s. A data cube would typically provide access to precomputed, summarized business or statistics data over multiple dimensions via OLAP queries. (Han et al., 2012, p. 187; Nativi et al., 2017, p. 76; Strobl et al., 2017, p. 32) *Earth Observation Data Cubes* (EODC), a term which was coined by Lewis et al. (2016) from Geoscience Australia, are a speciality in this regard because they are composed of earth observation data like satellite imagery. Individual geo-referenced satellite tiles within a specified time range are stacked in a way that they form a pixel-aligned image cube (see Fig. 10).

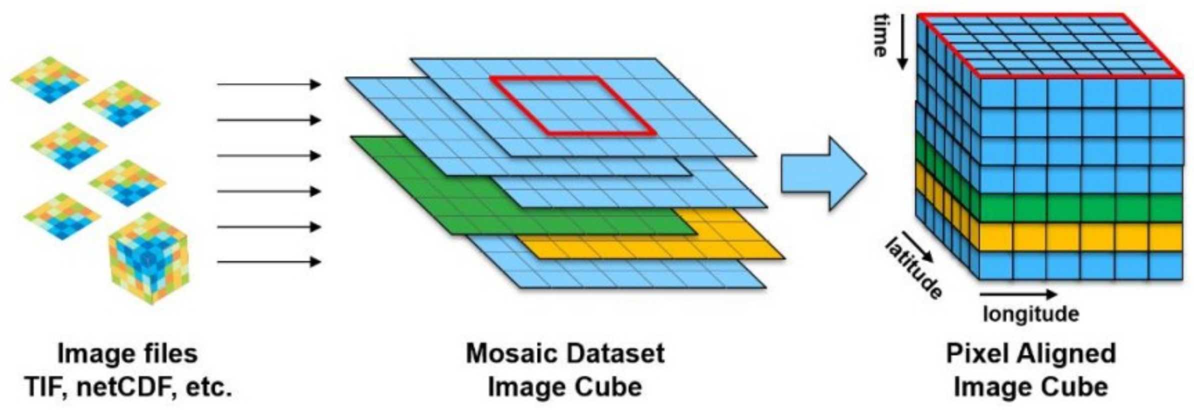


Fig. 10: Process from image files to a pixel-aligned image cube. Source: (Kopp et al., 2019, p. 8), cropped

Also sometimes called *Big Earth Data Cubes* (Baumann, 2017b; Nativi et al., 2017) or *Geospatial Data Cubes* (Strobl et al., 2017, p. 32), an EODC is defined by (Baumann, 2017a) as

> "a massive multi-dimensional array, also called "raster data" or "gridded data"; "massive" entails that we talk about sizes significantly beyond the main memory resources of the server hardware.

Data values [...] sit at grid points as defined by the *d* axes of the *d*-dimensional datacube. Coordinates along these axes allow addressing data values unambiguously."

This definition is part of the often-cited (Augustin et al., 2019, p. 1; Giuliani et al., 2017, p. 102; Giuliani, Masó, et al., 2019, p. 18; Purss et al., 2019, p. 64; Strobl et al., 2017, p. 32) *Datacube Manifesto* that also identifies six principles or requirements of data cube services: (Baumann, 2017a)

1. Regularly or irregularly gridded data; at least 1–4 dimensions (e.g. 1D sensor time series, 3D x/y/t image time series)
2. A single access pattern for all axes, irrespective of semantics (spatial, temporal, others)
3. Efficient trimming and slicing[17] along any number of axes in a single request
4. Similar access performance along any data cube axis
5. Invisible partitioning
6. A high-level query language "where users describe what they get, not the detailed algorithm"

In order to elaborate on the concepts of the *Datacube Manifesto* (Strobl et al., 2017) present *the six faces of the data cube*. Since a data cube must "allow ingestion, storage, provision, and analysis of structured geospatial data" (Strobl et al., 2017, p. 32) it has multiple technical aspects, one half of them data-oriented and the other half functionality-oriented (see Fig. 11).

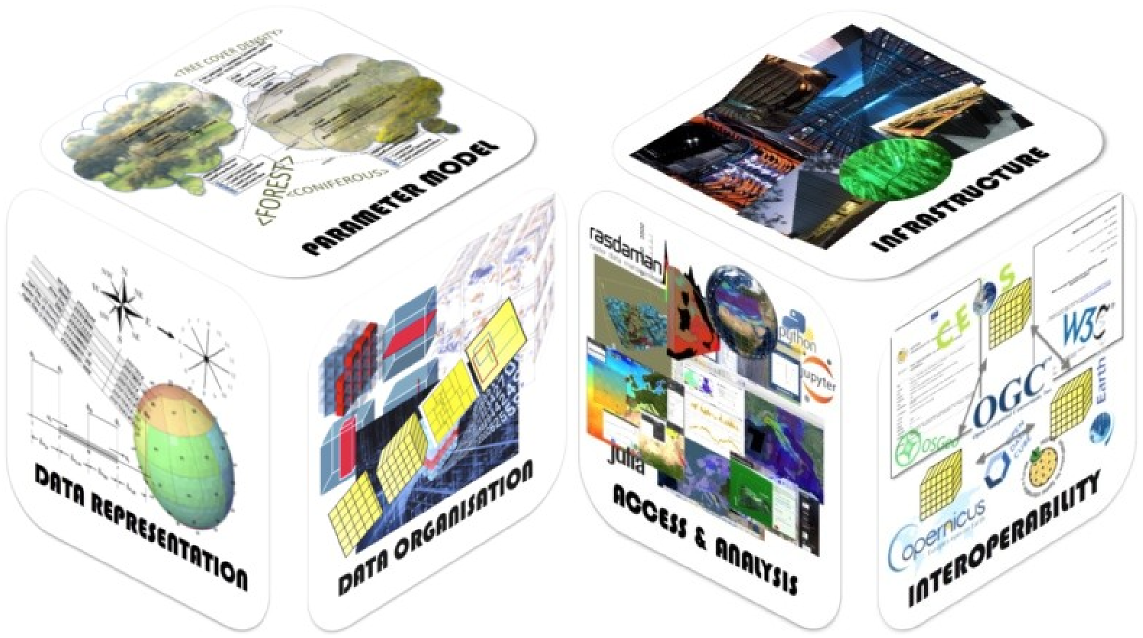


Fig. 11: Data-oriented faces (left) and functionality-oriented faces (right) of a data cube. Source: Strobl et al., 2017, p. 33f

These are the six *faces* (Strobl et al., 2017):

i. **Parameter Model:** Model that describes the semantics of a cube cell value. Challenges arise with incorporating data that describes the same parameter, but is of different origin or has been pre-processed differently.

ii. **Data Representation:** Discretization/Semantical encoding of a parameter, i.e. gridding of the spatial domain along a coordinate system

[17] Trimming → data cube subset with same dimensions, Slicing → data cube "slice" with lower dimensions (OGC, 2010, p. 3)

iii. **Data Organisation:** Actual arrangement and storage of the data cube: file format, file system, database structure. *Big Data* cubes additionally should deal with partitioning and streaming.

iv. **Infrastructure:** Storage and processing must be possible within the same IT infrastructure in a centralised or distributed fashion.

v. **Access and Analysis:** Interactive interfaces (APIs[18], GUIs[19]) must be provided for accessing, manipulating and analysing the data cube and its metadata. The data cube should provide anticipative processing cost estimations and handle access rights and security aspects.

vi. **Interoperability:** International standards should ensure communication among different data cube implementations to avoid silo effects. Different client software should be able to access data cube information independently from the implementation.

The *six faces of the data cube* have found resonance with (Nativi et al., 2017), who employ standardized software architecture modelling (*viewpoints modelling*) to them, with interoperability in mind. (Nativi et al., 2017, p. 79) By the use of this modelling they identify *concerns*, *stakeholders* and software design *patterns* for each of the *six faces*, summarised as six *views* (see Fig. 12). Nativi et al. (2017, p. 77) argue that the existing experience from OLAP and business intelligence can be reused and applied to EODCs. Therefore these existing technologies play an important role in their *views*.

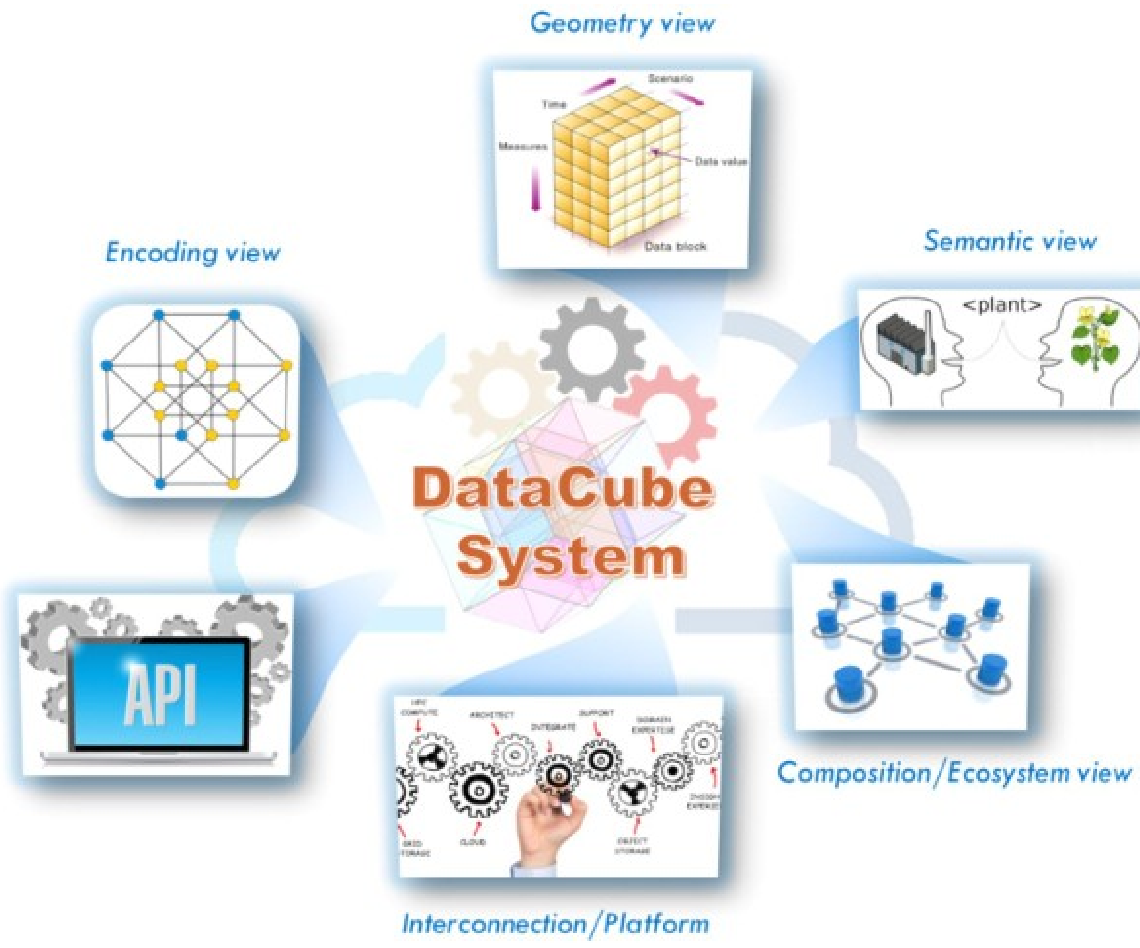


Fig. 12: The six interoperability views. Source: Nativi et al., 2017, p. 83

[18] API = Application Programming Interface
[19] GUI = Graphical User Interface

The six *interoperability views* are as follows:

- **i. Semantic view:** (Nativi et al., 2017, Table 1)
  - **a. Concerns:** analytical purpose of cube, information about cube's content, meta-data
  - **b. Stakeholders:** Earth system domain experts, community of practitioners and business case experts
  - **c. Patterns:** specifications (INSPIRE[20] data definition schemas, CF[21] convention, etc.), semantic and ontological languages (OWL[22], RDF[23], etc.)
- **ii. Geometry View:** (Nativi et al., 2017, Table 2)
  - a. **Concerns:** Geometrical representation and discretization of content, dimension neutrality (all dimensions should be treated the same)
  - b. **Stakeholders:** Business domain experts, geospatial information experts, business intelligence professionals
  - c. **Patterns:** OGC[24] feature and coverage general models, CDMs[25], OLAP modelling
- iii. **Encoding View:** (Nativi et al., 2017, Table 3)
  - a. **Concerns:** Multidimensional content encoding and storage (file formats, file systems, database structures, tiling strategy, etc.), pre-processing and query optimization
  - b. **Stakeholders:** Multidimensional data storage experts, OLAP experts
  - c. **Patterns:** File systems and formats (e.g. netCDF, HDF, GeoTIFF, GML, JSON, etc.; also see section 2.8), multidimensional databases, big data tiling strategies
- iv. **Interconnection/platform view**: (Nativi et al., 2017, Table 5)
  - a. **Concerns**: software components and services, system design and scalability, exposed APIs and control mechanisms
  - b. **Stakeholders:** Software engineers, HPC[26] infrastructure experts
  - c. **Patterns:** SoS[27] patterns, software design patterns (separation of concerns, transparency, reusability, decentralization), cloud computing interoperability patterns
- v. **Interaction/interface view:** (Nativi et al., 2017, Table 4)
  - a. **Concerns:** system functionality and accessibility to user, set of possible operations, web-based APIs
  - b. **Stakeholders:** data analysts, business intelligence professionals, web API experts, interoperability experts
  - c. **Patterns:** Web APIs (e.g. REST[28]), interactive notebooks (e.g. Jupyter notebooks), OLAP APIs, analytical languages (e.g. OGC WCPS[29])
- **vi. Composition/Ecosystem view:** (Nativi et al., 2017, Table 6)

[20] INSPIRE = Infrastructure for Spatial Information in the European Community
[21] CF = Climate and Forecast
[22] OWL = Web Ontology Language
[23] RDF = Resource Description Framework
[24] OGC = Open Geospatial Consortium
[25] CDM = Common Data Model
[26] HPC = High-Performance Computing
[27] SoS = System of Systems
[28] REST = Representational State Transfer
[29] WCPS = Web Coverage Processing Service

a. **Concerns:** system interoperability, distribution, scalability, governance
b. **Stakeholders:** SECO[30] and SoS experts, international standards experts, interoperability experts, system and policy managers, international organizations
c. **Patterns:** SECO patterns, DGGS[31], SoS architectures and governance styles

More recently, Augustin et al. (2019, p. 1f) use the term *view* differently and refer to EO data cubes as "logical views on EO data". *Logical* in this sense means that EO data is not accessed by file name, but via an API or a query language using spatio-temporal coordinates. From a technological perspective, they distinguish between *indexing* and *ingestion*: *Indexing* simply references the satellite scenes in the original data format and *ingestion* builds a multi-dimensional data structure, making time series or spatial analysis more efficient.

In the recent years EODC technologies have rapidly evolved and diversified. One approach have been Array Database Management Systems (ADBMS) such as RasDaMan, SciDB and TileDB. They represent EO data as multidimensional regular arrays and provide their own query languages and tiling mechanisms (chunking). Another approach has been to use distributed file systems such as Google File System (GFS) or Hadoop Distributed File System (HDFS). Their main advantage for performance is that data is processed on the same node as where it is stored. (Gomes et al., 2020, p. 2)

Still, these approaches do not meet nowadays' requirements anymore because "in an increasing number of cases, the volume is too large to move [EO] data to a local analysis platform". (Woodcock et al., 2016, p. 13) Thus, the *Moving Code* paradigm needs to be employed: Code must be shipped to the data instead of the other way round; moving data must be avoided. Cloud computing infrastructures like AWS or the European DIAS platforms (see section 2.4) form processing environments where the analyses can run on servers close to the data, improving performance. However, the platforms require high technical knowledge to operate; they supply satellite imagery in a file-based way and do not provide any abstracted interfaces to the data (Gomes et al., 2020, p. 2f)

Gomes et al. (2020, pp. 3, 14) identify seven platforms that conform to their definition of a *Platform for big EO Data Management and Analysis*. They define them

> "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."

The seven platforms include

- *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*,
- *Joint Research Center* (JRC) *Earth Observation Data and Processing Platform* (JEODPP) and
- *pipsCloud*.

[30] SECO = Software Ecosystem
[31] DGGS = Discrete Global Grid Systems

They then proceed with an evaluation on the basis of EO community needs that stem from related literature. (Gomes et al., 2020, pp. 14–21) Concluding, Gomes et al. (2020, p. 22) write that to meet all user needs a platform would have to provide two forms of processing: An abstracted API and access to low-level extensions. The latter form is already implemented in ODC and they "believe that ODC is the solution that presents the best conditions to evolve to a platform with these characteristics."

Since all of these platforms are relatively young, with GEE being the pioneer (GEE was launched in 2010), there is almost no empirical research on the performance of these solutions. One paper could be found by Wang et al. (2019) that reports performance measurements using the PIPS (Parallel Image Processing System) solution of the Chinese Academy of Sciences Institute of Remote Sensing and Digital Earth. (Wang et al., 2019, pp. 155–158) Wang et al. (2019, p. 167f) generated a Landsat-5 mosaic composed of 28 satellite images with seven bands over North-Eastern China (approx. 1,208,000 km^2). They used a multi-core cluster with up to 10 nodes with 8 Virtual CPUs and 16 GB memory each. When using only one node, a total runtime of over 350 minutes was recorded. At three nodes the runtime could be reduced to 100 minutes, while there was no further improvement from five or more nodes (about 80 minutes runtime). From ten nodes onwards the runtime increased again.

2.6 Euro Data Cube

Taking into account the previous chapter, the Euro Data Cube (EDC) is probably best introduced by its differences to the Open Data Cube. The CEO of EDC consortium's lead company *Sinergise* from Slovenia, Grega Milczinski (2020c), states that there is a difference in target audiences. ODC aims to "[g]ive scientists and other users easy ability to perform Exploratory Data Analysis", whereas EDC wants to "[m]ake it easy for developers to build tools" to aid scientists and other users with Exploratory Data Analysis. He highlights that ODC would primarily provide software and, in contrast, EDC focuses on making web services directly accessible to developers while also providing software for those who want to use it. (Milcinski, 2020c)

As mentioned above, the EDC consortium is lead by the Slovenian company *Sinergise* and includes *Brockmann Consult* from Germany, *EOX* (Austria) and *gisat* (Czech Republic). (EDC Consortium, 2020a) The consortium was founded in response to an ESA project tender for an *EO Data Cube Facility Service* with a project time frame of five years (2018–2022). (ESA, 2018b, p. 1) The Statement of Work (SOW) for that tender describes in detail the background and required tasks of the project. A paradigm shift from "EO Data Repository" to "EO Information Factory" forms the context of the tender: Instead of downloading massive amounts of satellite data to their local environment, users should be able to analyse and exploit the data directly where they are stored. (ESA, 2018b, p. 7) By adopting standard data formats and tools that are non-exclusive to earth observation, ESA aims to enlarge the user base of EO data from the space community to the larger GIS community. Finally, open source software and interoperability standards should remove barriers between (space and non-space) data sets, sensor types and data cube implementations. (ESA, 2018b, p. 7ff)

To this end ESA formulated six "Use Case Scenarios and Service Requirements" (ESA, 2018b, p. 15):

- **Information Layer Publishing & Marketplace:** Customers of the data cube should be able to publish their own thematic layers and make them available under commercial or non-commercial licences, generating income in the former case. (ESA, 2018b, p. 15f)

- **On-Demand Mapping:** Instead of data downloads or storage duplication, customers should be able to plug in their algorithm and run it directly on the latest source satellite imagery within the cloud environment. (ESA, 2018b, p. 16f)
- **Cross-Mission Analysis:** The data cube should rely on satellite data already provisioned in the cloud and avoid duplication because of the high storage costs of the latter scenario. Customers should be able to choose resolution, grid and projection dynamically and the data cube should harmonise the source data from different satellite missions on the fly. (ESA, 2018b, p. 17f)
- **Front-End Operator Support:** The data cube should provide interfaces that customers can use to build front-end services on top of the data cube engine. (ESA, 2018b, p. 19)
- **Virtual Thematic Data Cubes:** Through trimming and slicing customers should be able to limit the data cube's view and make only subsets available to their communities. (ESA, 2018b, p. 19)
- **Data Cube Federation & Interoperability:** The contractor should actively participate in the standardization process of the Open Geospatial Consortium (OGC). Interoperability with the Open Data Cube and with regional data cube instances ("national data on nationally operated infrastructure") should be ensured. (ESA, 2018b, p. 19f)

In order to fulfil these requirements the EDC consortium designed a data cube architecture comprised of multiple web services, as can be seen in Fig. 13. At the bottom the data sources are shown: Besides the Sentinel satellites' imagery, data from the Landsat-8 mission and the MODIS instrument (NASA[32]/USGS[33]), derived Copernicus layers (Copernicus Climate Change Service, Marine Service, Land Use Monitoring Service, Atmosphere Monitoring Service) and commercial satellite imagery (Airbus SPOT and Pléiades, PlanetScope), among others, are available through the data cube. The *bring-your-own-data* concept allows customers to link their own data to the data cube for combined analysis and publish it to a wider user community. (EDC Consortium, 2020b, p. 4f)

The cloud section in Fig. 13 contains the data repositories from where data can be obtained; options are the DIAS systems, AWS and customer-owned object stores. The Euro Data Cube services are built on top of that: (EDC Consortium, 2020b, pp. 4, 8–14)

- **Sentinel Hub:** A cloud-based REST API combined with custom JavaScript scripts for on-the-fly or batch processing of satellite data.
- **xcube:** Generation, analysis and publication of custom data cubes, which can be a combination of EO and non-EO data and/or gridded and non-gridded data
- **geoDB:** PostgreSQL with user access management to store feature data, use it for one's own computations and publish it non-commercially or commercially
- **EOxHub:** The workspace combines a dashboard, a Jupyter notebooks computing environment and a marketplace. The latter is a repository where customers can unlock additional computing resources, buy the aforementioned web services and share/sell web apps and algorithms to other customers.

[32] NASA = National Aeronautics and Space Administration
[33] USGS = United States Geological Survey

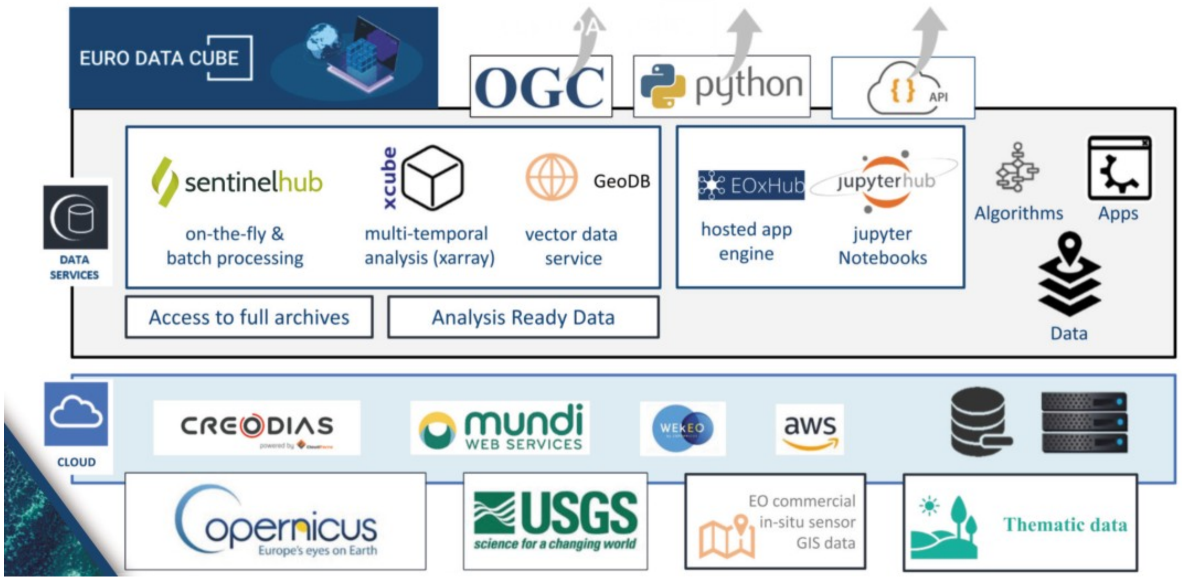


Fig. 13: Euro Data Cube architecture. Source: EDC Consortium, 2020b, p. 4

The Sentinel Hub Batch Processing API is of high relevance for this master's thesis and its technical details will be discussed below. A prerequisite for using the EDC is an active EOxHub workspace subscription starting from EUR 99 per month for up to 6 CPUs and up to 24 GB Memory. (EDC Consortium, 2020b, p. 6f) Processing costs with the Sentinel Hub are tracked via an abstract currency called *processing units* (PUs), which can be obtained with real money. One PU is defined as the cost for "an output (image) size of 512 x 512 pixels, 3 dataset input bands, one data `sample` per pixel [...], an output (image) format not exceeding 16 bits per pixel, without additional processing (e.g. orthorectification) applied." (Sentinel Hub, 2020c) A *sample* is a particularity of a processing script and described below.

There are subscription plans as well as pre-paid plans available for the Sentinel Hub. The Exploration plan for non-commercial use is EUR 30 per month and includes 30,000 processing units. Commercial plans offer 70,000 PUs for EUR 100 monthly, 400,000 PUs for EUR 500 and 1,000,000 PUs for EUR 1,000. The pre-paid version of the Sentinel Hub subscription sells 1,000 PUs (valid 24 months) for EUR 2.50 for the first 400,000 PUs and EUR 1.50 for any additional PUs. Subscription plans similarly can be topped-up for EUR 1.50 per 1,000 PUs.

2.6.1 EDC Batch Processing API

The Batch Processing (BP) API is an asynchronous REST API, meaning that results are not returned in an immediate response, but delivered to a customer-owned S3 bucket. This makes sense for large processing requests that take longer than a few minutes because it is impractical to maintain a service connection until the request finishes. The area of interest of these large requests may span an entire country or continent and therefore—in order to optimize execution time—they are processed in parallel in a distributed fashion. The possible workflows for such an asynchronous request are shown in Fig. 14. Using the HTTP[34] method *POST* a new request can be created and commands like *START*, *ANALYSE* or *CANCEL* can be issued to the service. The request's current status can be queried via a HTTP *GET* call. (Milcinski, 2020b; Sentinel Hub, 2020b)

[34] HTTP = Hypertext Transfer Protocol

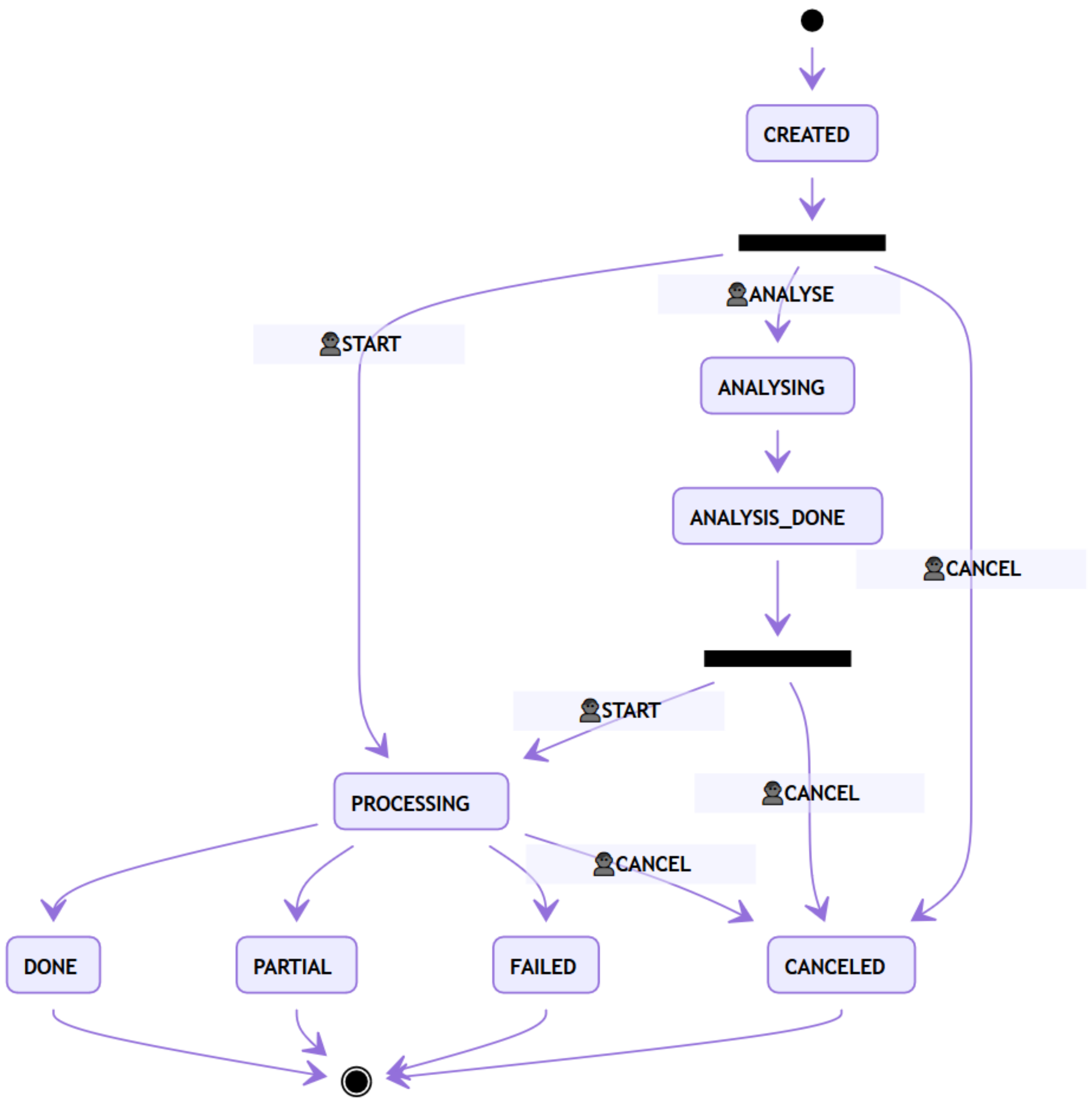
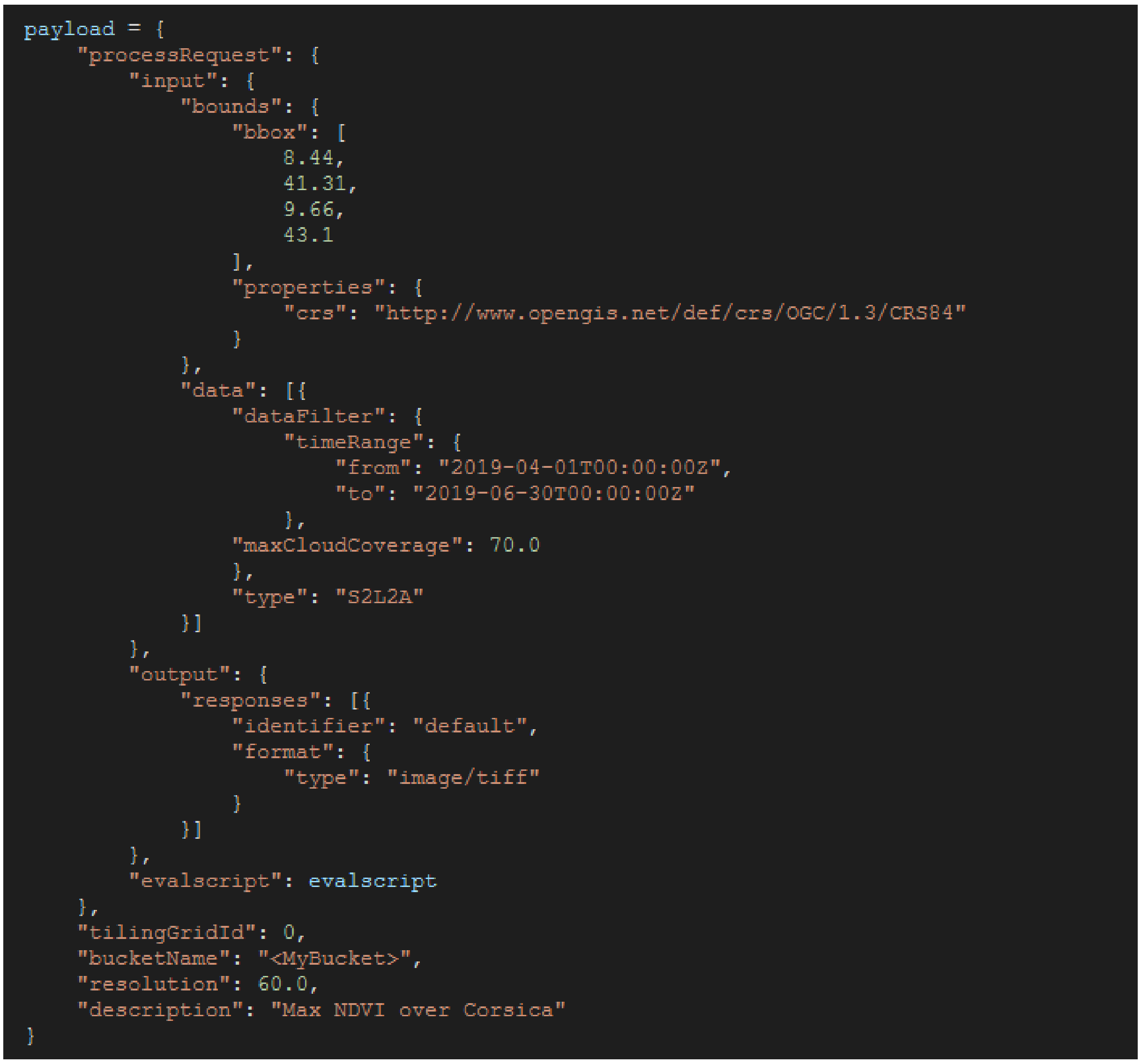


Fig. 14: EDC Batch Processing API asynchronous workflow. Source: Sentinel Hub, 2020b

After a processing request has been created an initial estimate for the *processing units* is provided when the request's status is queried. This estimate can be refined by *POST*ing *ANALYSE* to the request's API endpoint. The command *START* moves the request into the *PROCESSING* state, which is resolved after "5 minutes to a couple of hours" (Milcinski, 2020b) to *DONE* or—if some or all tiles have failed—to *PARTIAL* or *FAILED*, respectively. A request can be cancelled at any time during the workflow by issuing the command *CANCEL*. (Sentinel Hub, 2020b)



```
payload = {
    "processRequest": {
        "input": {
            "bounds": {
                "bbox": [
                    8.44,
                    41.31,
                    9.66,
                    43.1
                ],
                "properties": {
                    "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
                }
            },
            "data": [{
                "dataFilter": {
                    "timeRange": {
                        "from": "2019-04-01T00:00:00Z",
                        "to": "2019-06-30T00:00:00Z"
                    },
                "maxCloudCoverage": 70.0
                },
                "type": "S2L2A"
            }]
        },
        "output": {
            "responses": [{
                "identifier": "default",
                "format": {
                    "type": "image/tiff"
                }
            }]
        },
        "evalscript": evalscript
    },
    "tilingGridId": 0,
    "bucketName": "<MyBucket>",
    "resolution": 60.0,
    "description": "Max NDVI over Corsica"
}
```

Fig. 15: Example JSON payload that is attached to a Batch Processing request. Source: (Sentinel Hub, 2020e)

In order to create a request a JSON[35] payload (see Fig. 15) must be attached to the HTTP *POST* call. The payload lets the user specify input parameters like the area of interest, the time range, the output S3 bucket and the tiling grid to be used. Three pre-defined grids are available, with differing spatial resolutions and tile extents (see Table 4). The 10-km and 100.08-km grids are self-explanatory; the "s2gm" grid was designed during Sentinel Hub's Sentinel-2 Global Mosaic (S2GM**)** project, mainly to avoid too many overlaps. All grids are based on the UTM projection. (Milcinski, 2020b)

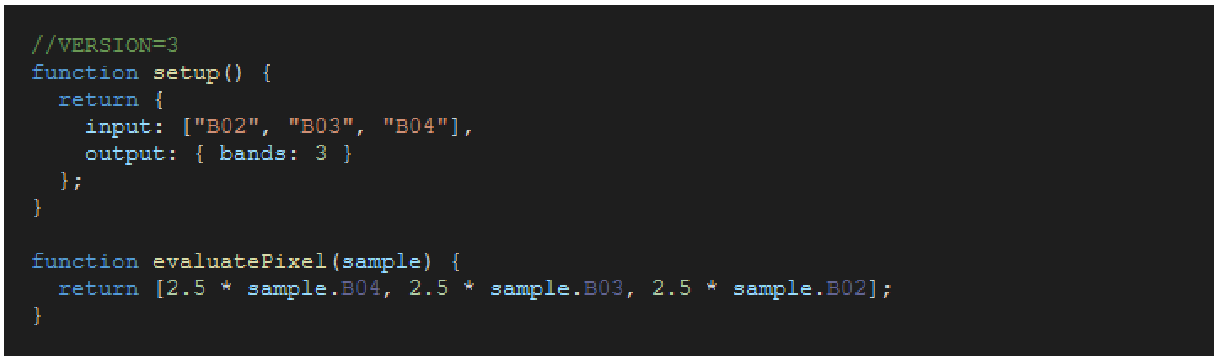
Name	s2gm grid	10-km grid	100.08-km grid
Id	0	1	2
tileWidth [m]	20,040	10,000	100,080
tileHeight [m]	20,040	10,000	100,080
resolutions [m]	10.0, 20.0, 60.0	10.0, 20.0	60.0, 120.0, 240.0, 360.0

Table 4: Sentinel Hub Batch API tiling grids. Source: Sentinel Hub, 2020b

The JSON payload also includes a so-called *evalscript*. An evalscript is composed of two mandatory functions: The *setup* function defines the input bands and output format of the image(s) and the *evaluatePixel* function contains the actual processing instructions. An example is shown in Fig. 16:

[35] JSON = JavaScript Object Notation

Satellite bands 2 (blue), 3 (green) and 4 (red) are fed into the process and the values are augmented by a factor of 2.5. A three-band true-colour image of the requested area of interest is delivered to the defined S3 bucket. (Sentinel Hub, 2020d)



```
//VERSION=3
function setup() {
  return {
    input: ["B02", "B03", "B04"],
    output: { bands: 3 }
  };
}

function evaluatePixel(sample) {
  return [2.5 * sample.B04, 2.5 * sample.B03, 2.5 * sample.B02];
}
```

Fig. 16: Example of a simple Batch Processing evalscript. Source: Sentinel Hub, 2020d

The *setup* function must return an object that has band names as *input* properties and one or multiple output objects each with a user-defined *id* (which becomes the file name, optional), a number of bands of the respective output image file and a *sampleType* (optional). The *sampleType* describes the data type of the output image file and can be one of *UINT8* (256 distinguishable values), *UINT16* (65,536) or *FLOAT32* (decimals between -2^{127} and 2^{127}). Besides, the *setup* function's return object can contain a *mosaicking* specifier that defines which *samples* are taken into account for the processing. *mosaicking* can either be set to *SIMPLE* (only one *sample* for the complete requested time range is evaluated), *ORBIT* (one *sample* per orbit within the requested time range) or *TILE* (all *samples* within the requested time range, possibly from multiple scenes in the same orbit). (Sentinel Hub, 2020d)

The second mandatory function is called *evaluatePixel*. Inside the function one or more *sample(s)* are available, depending on the *mosaicking* type chosen in *setup*. One or more *scene(s)* object(s) carry metadata information including the date(s), original name(s) of the satellite image(s) and *orbitId(s)* belonging to the sample(s). Some additional (meta)data can be passed on to *evaluatePixel* via the *inputMetadata*, *customData* and *outputMetadata* arguments. If there is only one output image defined, the function must return a value for each band of that image. For multiple output images values must be assigned to the respective *id* that has been set in the *setup* function. (Sentinel Hub, 2020d)

As long as the request's status is *PROCESSING,* the results start appearing in the S3 bucket that the user specifies. The default folder structure can be seen in Table 5. For every request a new directory is created and a JSON file with the request parameters is uploaded. Parallel processing requires a partitioning of the area of interest into several tiles, which are processed independently from each other. Therefore the bucket will contain a folder for each tile that in turn contains the requested output images as JPEG[36], PNG[37] or Cloud-optimized GeoTIFF[38] (COG); (Sentinel Hub, 2020b) the processing result is not merged to one single file.

[36] JPEG = Joint Photographic Experts Group
[37] PNG = Portable Network Graphics
[38] TIFF = Tagged Image File Format

Request ID	Tile ID	Image files
05baa5b2-3dee-43fe-8e67-b8195b095ed0/	11375/	rgb.tif false-color.tif …
	11376/	rgb.tif false-color.tif …
	11377/	rgb.tif false-color.tif …
	…	…
	request.json	

Table 5: Illustrative folder structure of Batch Processing API results inside an S3 bucket. Source: own work

2.7 mapchete Hub

mapchete Hub is a cloud processing service for satellite imagery and other geodata written in Python. It extends the open-source software library *mapchete* with the libraries *Celery* and *MongoDB* into an asynchronous REST API that can be controlled via a command-line tool called *mhub*. The mapchete Hub code is developed and maintained by Joachim Ungar with contributions from Petr Ševčík (both EOX IT Services GmbH). (EOX, 2020a)

Ungar is also the creator of the name-giving mapchete Python library itself that takes large amounts of input data, chunks it into smaller parts (*tiles*) and processes them, several at a time. Optionally—as mapchete was designed for web maps—a *tile pyramid* in the Web Map Tile Service (WMTS) format can be requested as output format (see Fig. 17). A tile pyramid consists of multiple zoom levels: For example, the Web Mercator projection zoom level 0 covers the whole world on one single tile while level 1 covers the world on four tiles, and so on. Each zoom level has its own tile matrix and individual tiles are referenced by their zoom level, their row and their column. Besides the predefined tile pyramids for the WGS84 and Web Mercator projections, users can define their own custom tile pyramids for any other existing projection. (EOX, 2020f, 2015/2020)

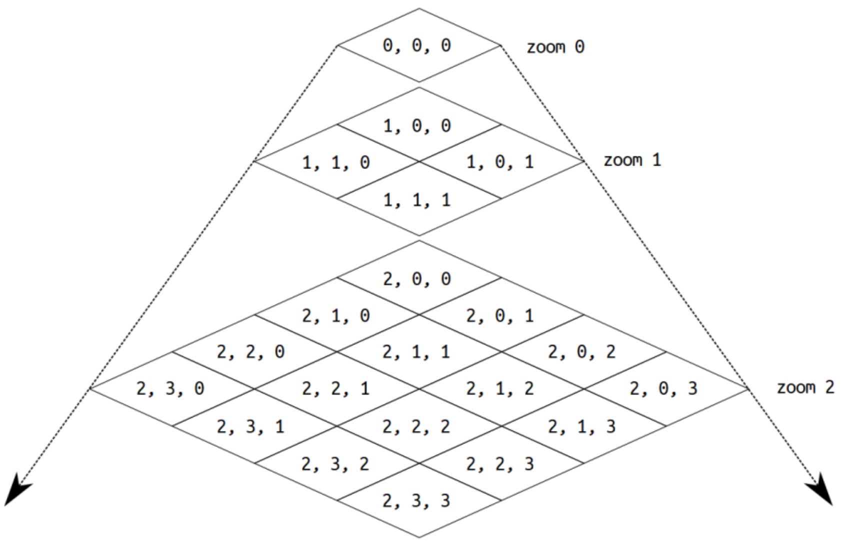


Fig. 17: Web Mercator tile pyramid. Source: EOX, 2020f

Per definition, each WMTS tile has a pixel size of 256 x 256 pixels. (Masó, 2016) For processing, the small tile size can mean some unnecessary computational overhead. In order to provide more efficient processing, mapchete employs a concept known as *metatiling*. Instead of running a process on multiple smaller tiles, tiles are first united to larger metatiles. mapchete's metatiling parameter can be set to 2, 4, 8 or 16, resulting in tile sizes of 512, 1,024, 2,048 or 4,096 pixels squared, respectively. (EOX, 2020f, 2020b)

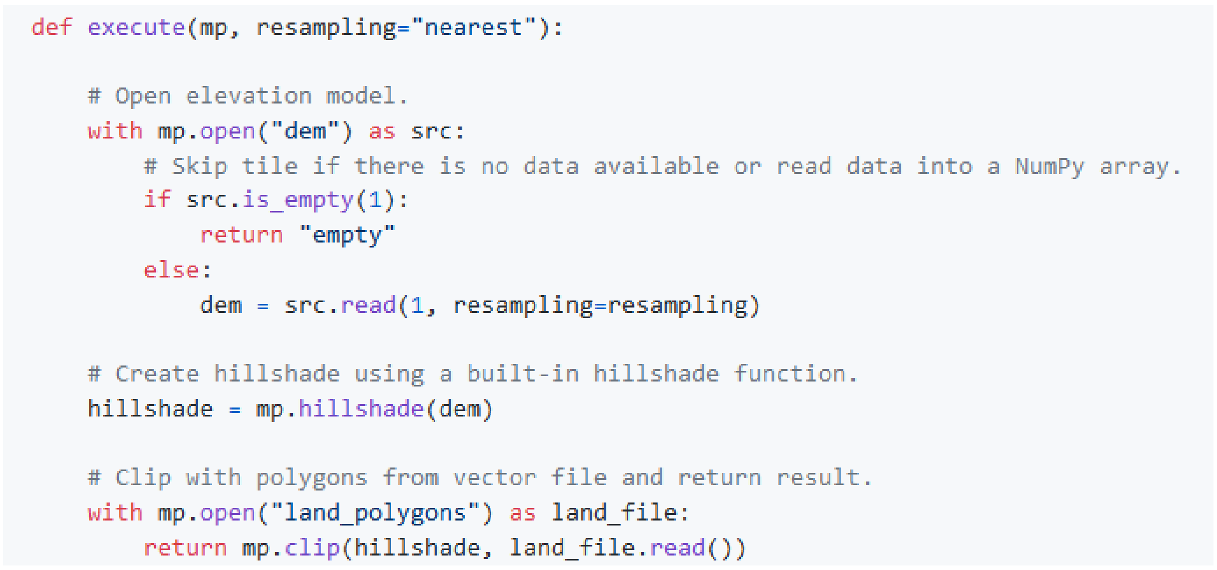
Configuration of mapchete is done using a Python process file and a YAML[39] configuration file that must have a *.mapchete* file extension. Apart from the pyramid definition, the *.mapchete* file makes it possible to specify input data and output format. An example can be seen in Fig. 18.

```yaml
process: my_python_process.py  # or a Python module path: mypythonpackage.myprocess
zoom_levels:
    min: 0
    max: 12
input:
    dem: /path/to/dem.tif
    land_polygons: /path/to/polygon/file.geojson
output:
    format: PNG_hillshade
    path: /output/path
pyramid:
    grid: mercator

# process specific parameters
resampling: cubic_spline
```

Fig. 18: Example for a *.mapchete* YAML file. Source: EOX, 2015/2020

The actual processing script must be made available to mapchete as a separate Python file. It must include an *execute* function, which receives an *mp* object and, optionally, one or multiple custom variables. The *mp* object exposes the process parameters and offers various functions to open and manipulate the input data that is referenced under the *input* key in the configuration file. Custom variables can be defined in the *.mapchete* configuration file and are used to overwrite default values that the user can specify in the *execute* function (e.g. the *resampling* parameter in Fig. 18 and Fig. 19). After processing, the *execute* function must return one or more multi-dimensional arrays. (EOX, 2020c)

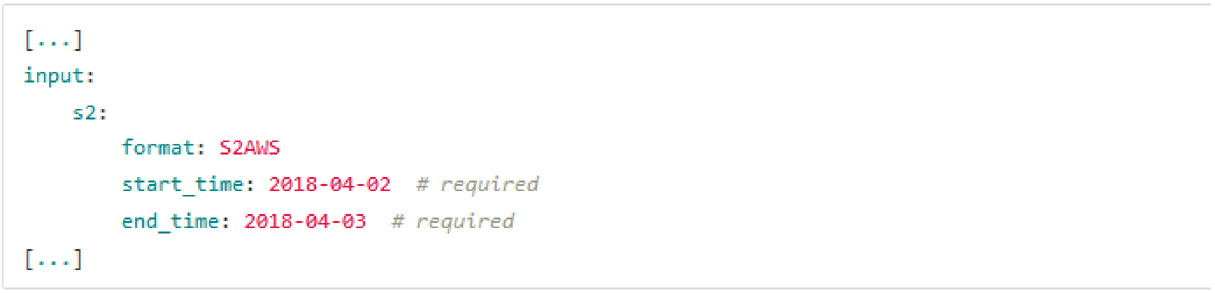
[39] YAML = YAML Ain't Markup Language



```python
def execute(mp, resampling="nearest"):

    # Open elevation model.
    with mp.open("dem") as src:
        # Skip tile if there is no data available or read data into a NumPy array.
        if src.is_empty(1):
            return "empty"
        else:
            dem = src.read(1, resampling=resampling)

    # Create hillshade using a built-in hillshade function.
    hillshade = mp.hillshade(dem)

    # Clip with polygons from vector file and return result.
    with mp.open("land_polygons") as land_file:
        return mp.clip(hillshade, land_file.read())
```

Fig. 19: Example of a simple mapchete processing script. Source: EOX, 2015/2020

mapchete cannot out-of-the-box use satellite imagery stored in the cloud. An input plug-in called *mapchete Satellite* makes this functionality available via additional configuration parameters. Specifically, mapchete Satellite provides access to Sentinel-1, Sentinel-2 and MODIS archives stored in the AWS and Mundi DIAS clouds. The parameters that must at least be added to the *.mapchete* file are the *start_time* and *end_time* of the time stack that is to be processed (see Fig. 20). (EOX, 2020d)

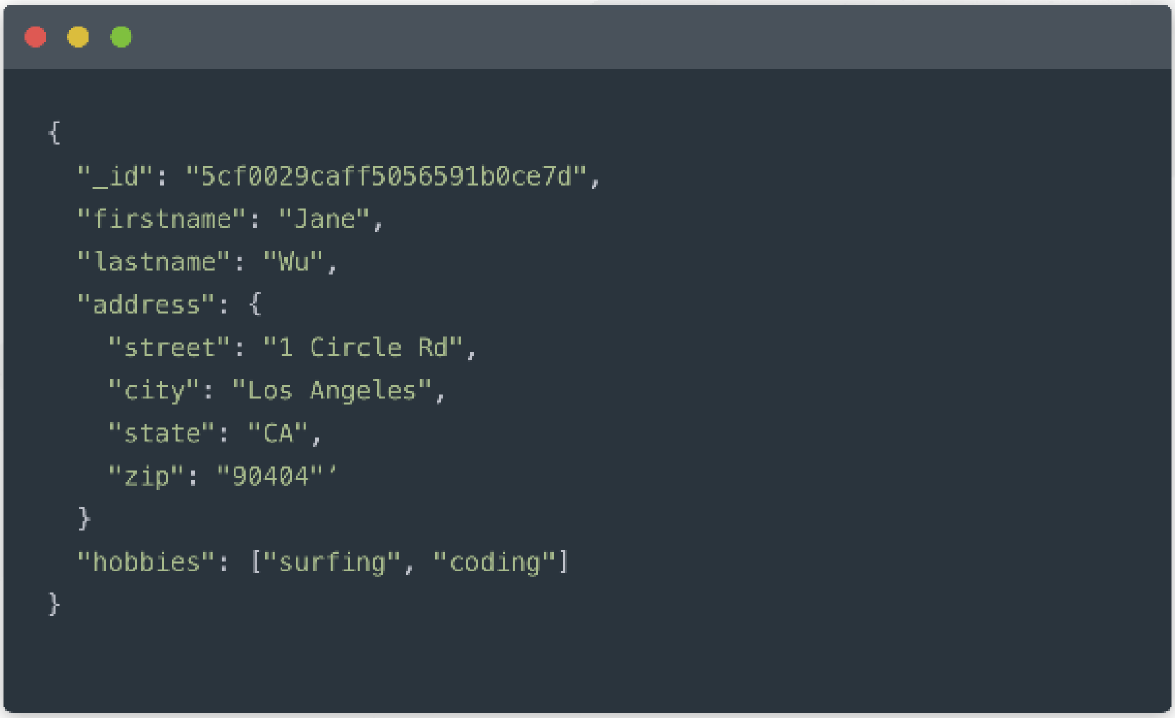


```yaml
[...]
input:
    s2:
        format: S2AWS
        start_time: 2018-04-02  # required
        end_time: 2018-04-03  # required
[...]
```

Fig. 20: Example of .mapchete file input section for satellite imagery from a cloud repository. Source: EOX, 2020d

mapchete Hub processing requests (or *jobs*) are managed by the asynchronous job queue library *Celery*. A queue is a structure that handles objects in a first-in/first-out manner. *Celery* takes over the task to assigns jobs to *workers*. Whenever a job is finished, it leaves the queue and *Celery* allocates the now unoccupied worker to the next pending job in the queue. (Celery, 2020b; Garner, 2020)

In order for this workflow to function properly, *Celery* needs a service to receive and send messages and a database where it can store the current state of a job. (Celery, 2020a) This is handled by mapchete Hub using *MongoDB*. (EOX, 2020a) According to their website, "MongoDB is a general purpose, document-based, distributed database". (MongoDB, 2020b) *Document-based* in this sense means that data is not stored as tables like in traditional relational databases. Instead, *MongoDB* uses a JSON-like object representation that is more familiar to developers and optimized for distributing documents across multiple servers in the cloud. (MongoDB, 2020a) Fig. 21 shows an example document reproduced as a JSON.



```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"'
  }
  "hobbies": ["surfing", "coding"]
}
```

Fig. 21: MongoDB document example reproduced as JSON. Source: MongoDB, 2020b

Putting it all together, a mapchete Hub instance is a combination of multiple services resulting in an asynchronous pendent to *mapchete*. The code base contains scripts for the CLI (command-line interface) that help start the required services in the cloud. For users the command-line utility *mhub* is provided that follows mapchete's CLI syntax and features additional functionality for *mapchete Hub*'s asynchronous nature. Using *mhub* it is possible to list all *jobs*, *execute* or *cancel* jobs and query a job's *status*, among several other commands. (EOX, 2020a)

The results of a processing job are stored alongside a *metadata.json* file in a chosen directory on the user's S3 bucket as COG, PNG or GeoJSON (for vector data) (EOX, 2020e) or as multi-dimensional arrays in the netCDF or *zarr* formats. For the latter option the open-source plug-in *mapchete_xarray* must be installed. (EOX, 2019/2020) An example output directory structure can be seen in Table 6.

User-defined directory	**Zoom level**	**Tile matrix row**	**Tile matrix column**
data_cube/	5/	800/	60.zarr 61.zarr 62.zarr 63.zarr …
		801/	60.zarr 61.zarr 62.zarr 63.zarr …
		…	…
	metadata.json		

Table 6: Illustrative folder structure of Mapchete Hub results inside an S3 bucket. Source: own work

2.8 Data representation

Several file formats have been discussed in the previous chapters like COG or netCDF. Generally, one has to distinguish between in-memory data representation and persistent file storage. While COG or netCDF are used to persist data in a file system, in-memory data is being used during computations. Software packages can transform data from one state into the other.

Most programming languages have simple structures in place to form collections of data, e.g. *arrays* in JavaScript or *lists* in Python. They can store simple data types like numbers or text together in one variable and their notation would be as follows:

```
list_or_array = [1, “two”, 3.4, etc]
```

In contrast to JavaScript, Python also features some more sophisticated data structures that are added through plug-in libraries: *NumPy* arrays, *pandas Series* and *DataFrame*, as well as *xarray DataArray* and *DataSet*. All of them offer unique benefits that might be speed improvements or labelled data tables to make data handling easier.

NumPy (short for *Numerical Python*) introduces n-dimensional homogeneous arrays to Python. *Homogeneous* in this regard means that a NumPy array only contains data of the same data type (e.g. only integers). This allows for faster mathematical computations and much less memory consumption. NumPy arrays can represent what would usually be called a vector (one-dimensional), a matrix (two-dimensional) or any higher-dimensional grid. (The SciPy community, 2020) An example of a three-dimensional NumPy array with a 16-bit integer data type can be seen here:

```
>>> np.ones( (2,3,4), dtype=np.int16 )
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
```

pandas extends on NumPy functionality by attaching an index and labels to vectors and matrices. Vectors correspond to pandas Series, where each vector entry has an index value attached. DataFrames mimic spreadsheets and can have both an index and column labels. Examples are shown below: (the pandas development team, 2020)

```
pd.Series(np.random.randn(5),    in-
dex=['a', 'b', 'c', 'd', 'e'])
a    0.469112
b   -0.282863
c   -1.509059
d   -1.135632
e    1.212112
dtype: float64
```

```
pd.DataFrame(data,     columns=['a',
'b'])
   a   b
0  1   2
1  5  10
```

xarray combines the labels from pandas and the performance benefits of NumPy’s n-dimensional arrays into DataArrays that can further be collected into DataSets. DataArrays do not necessarily have a uniformly-spaced coordinate system. Instead arbitrary *coordinates* can be specified, like points in time or text coordinates. The *dimensions* are like the column labels in pandas and describe the coordinates semantically. Additionally, custom metadata (so-called *attributes*) can be attached to a DataArray. A DataSet is a container for DataArrays and “is designed as an in-memory representation of the data model from the netCDF file format.” (xarray Developers, 2020a) A sample DataArray is reproduced here:

```
xr.DataArray(data, coords=[times, locs], dims=["time", "space"])
<xarray.DataArray (time: 4, space: 3)>
array([[0.127, 0.967, 0.26 ],
       [0.897, 0.377, 0.336],
       [0.451, 0.84 , 0.123],
       [0.543, 0.373, 0.448]])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-01 2000-01-02 2000-01-03 2000-01-04
  * space    (space) <U2 'IA' 'IL' 'IN'
```

According to its maintainer Unidata (2018), the netCDF[40] file format “is a community standard for sharing scientific data.” Among others, they highlight these features:

- **Self-describing:** Metadata is built into the file
- **Scalable:** Users can access small parts of a large dataset in the cloud
- **Appendable:** Data can later be appended to an existing netCDF file
- **Sharable:** „One writer and multiple readers may simultaneously access the same netCDF file” (Unidata, 2018)

The file format’s latest iteration netCDF-4 is based on the HDF5[41] format, which promises optimized performance through chunking (structuring data in multiple chunks for subset access or parallel computation), compression and encryption. (HDF Group, 2017; Unidata, 2020) There is another file format (and library) that provides similar functionality to HDF5—but claiming more flexibility—called *zarr*. While facilitating chunked and compressed NumPy arrays in-memory, zarr can also persist them on a local file system or in an object storage in the cloud. (Zarr Developers, 2019, 2020) xarray can leverage this zarr feature and store DataArrays and DataSets as zarr files. (xarray Developers, 2020b)

The last relevant storage format to be discussed in this thesis is Cloud Optimized GeoTIFF (COG). Despite sticking to the original GeoTIFF specification, COG imagery files are structured in a way that makes them more quickly accessible over the internet. This is achieved through *tiling* and *overviews*. Tiles are the imagery equivalent of chunks and allow file access in portions. Overviews are representations of an image with a lower resolution. They increase storage size of the file, but enable fast display of images when zoomed out. Tiling and overviews are combined with HTTP *GET* range requests, which make it possible that files serve only parts of themselves to the user. (cogeotiff, 2020)

[40] netCDF = Network Common Data Form
[41] HDF = Hierarchical Data Format

A recent study by Yee et al. (2020) compared the performance of COG, netCDF and zarr for Earth observation data. They found that, in general, COG is faster than zarr and netCDF has the poorest performance when it comes to various analysis operations (see Fig. 22; details to be gathered from the original study).

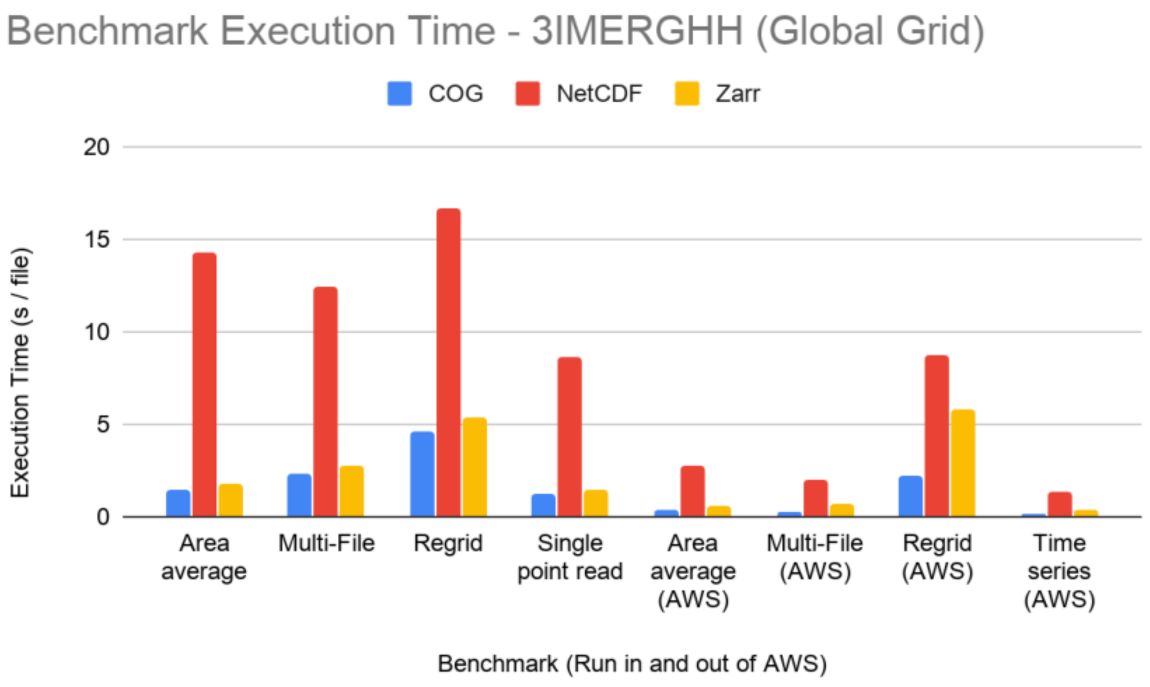


Fig. 22: Chart showing performance benchmark of COG, netCDF and zarr. Source: Yee et al., 2020, p. 18

netCDF also doesn't perform well when the percentage of the file that has to be read for an operation is evaluated. In half of the cases zarr beats COG and vice-versa (see Fig. 23).

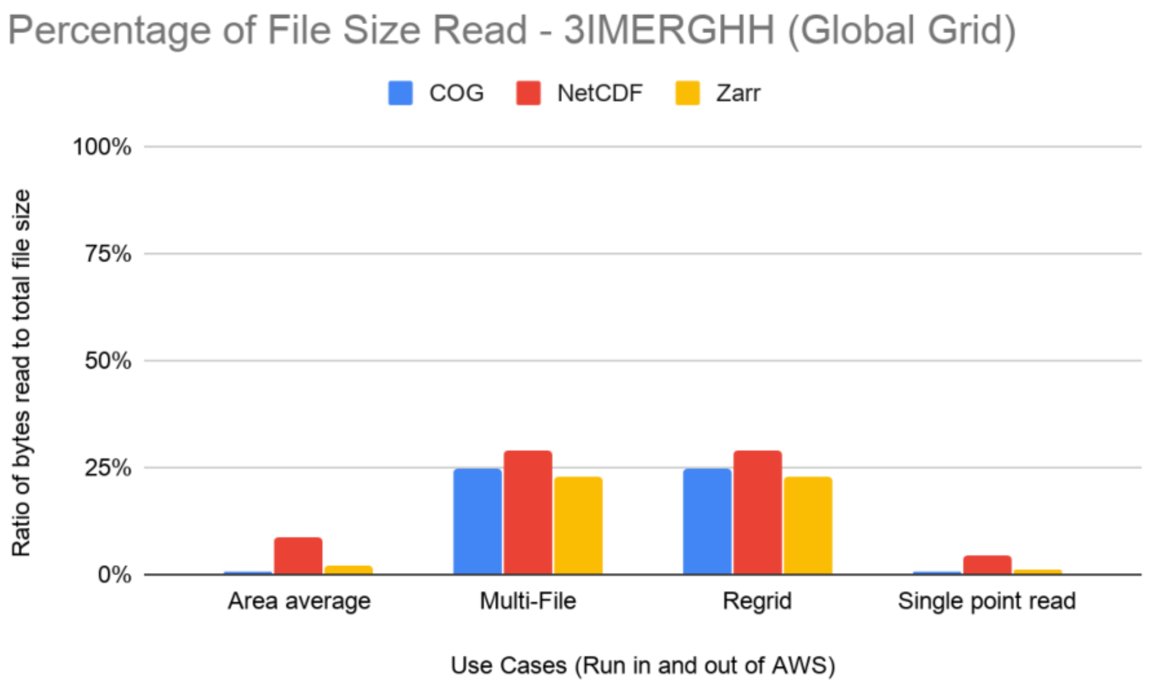


Fig. 23: Chart showing file size read benchmark of COG, netCDF and zarr. Source: Yee et al., 2020, p. 20

Yee et al. (2020, p. 23) also compared the file size of four datasets in different formats and the zarr version had a bigger storage footprint than netCDF in two of the three cases where it was tested. The authors of the study, however, note that this would be due to zarr's default compression settings which are optimized for certain use cases and lead to feeble results in others. The file size comparison is shown in Fig. 24.

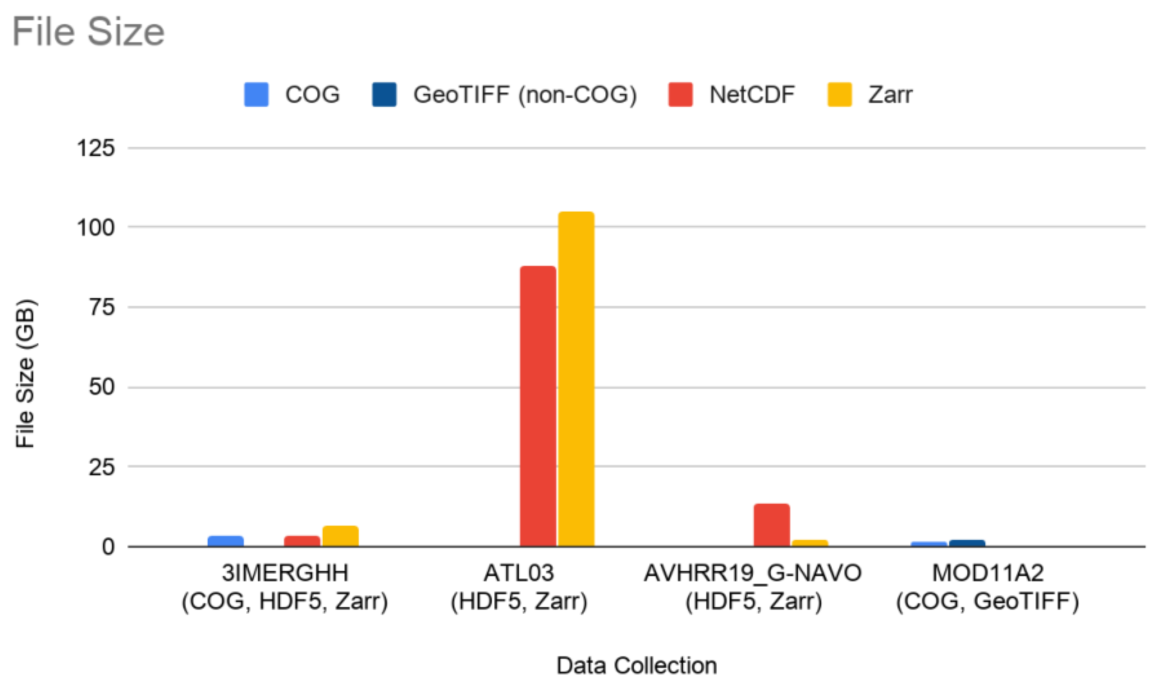


Fig. 24: Chart showing file size comparison of COG, GeoTIFF (non-COG), netCDF and zarr. Source: Yee et al., 2020, p. 23

In their final remarks Yee et al. (2020, p. 24f) emphasize that file packaging (e.g. how data is structured inside a file) highly impacts performance—even more than the choice of format. In respect to Earth observation data cubes, developers face a trade-off between favouring temporal analysis or spatial analysis: For a fixed chunk size, temporal analysis is more performant if a chunk covers a small spatial area, and instead contains a longer time series. Contrary to this, spatial analysis over larger areas would be more efficient if a chunk includes a wider area rather than more information on the time axis. Therefore chunking should be matched to the targeted use case.

3 Methodology

The following sections explain how the knowledge from chapter 0 was applied to the research problem at hand. Initially, details about the measurements of the performance metrics are disclosed. Then the test area is located and described, including the tiling approaches of BP and mHub. Since EDC xcube could not be used for such large areas itself, some properties of its data cube schema are copied and the resulting schemata are presented. Subsequently, the processing scripts are explained in detail. Finally, a chapter about the used cloud hardware gives insights into the computing resources available to mapchete Hub.

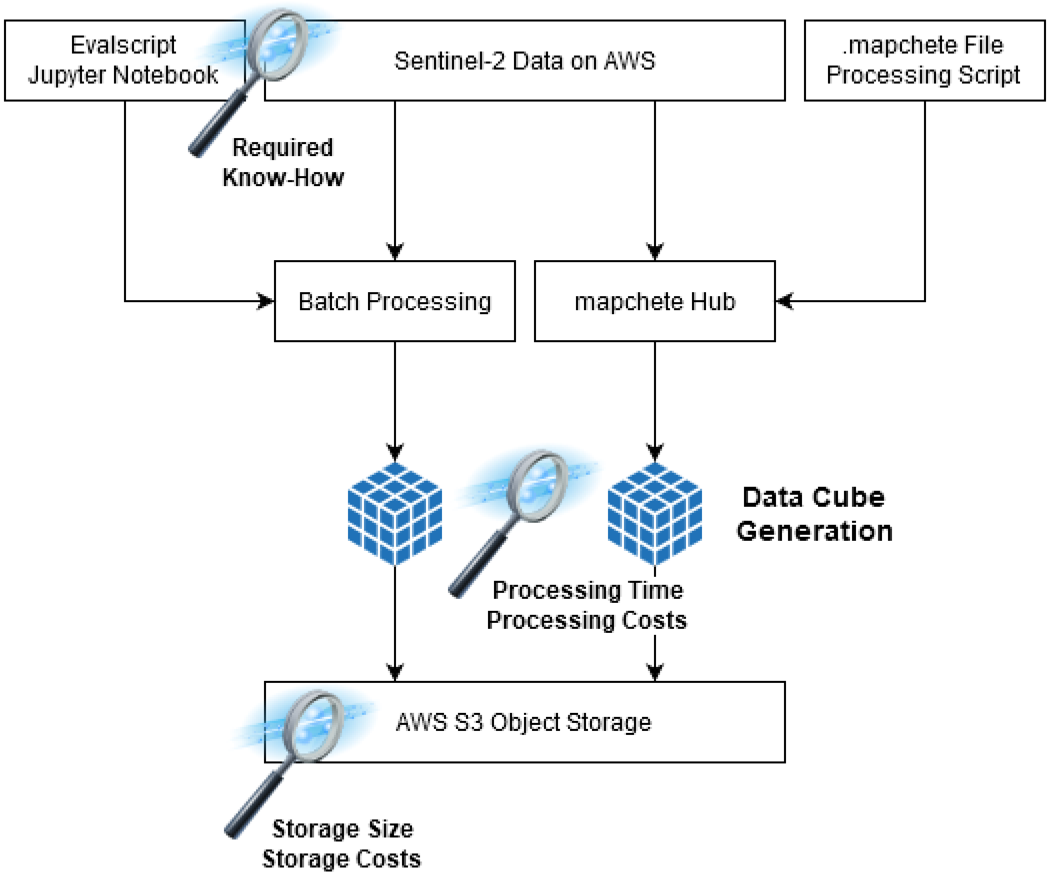


Fig. 25: Overview of the experiment workflow. Source: Own work

Fig. 25 shows an overview of the experiment workflow, from the satellite data repository on AWS via the two investigated applications to the resulting data cubes in the AWS S3 bucket. In general, the experiments were first conducted on EDC Batch Processing and then on mapchete Hub. In both cases the processing scripts were developed and tested with small areas and not used with the whole area of interest before they were working properly. A scale-up from short time ranges to the maximum of 8.5 months followed. The idea to conduct the experiments for only one month did not appear before the Sentinel Hub research account had already been closed down. Therefore results derived from longer time ranges via linear regression will be reported in chapter 0.

3.1 Performance metrics

Several metrics for computer performance measurements are of interest to make an informed business decision. First and foremost direct costs are an important factor, but also other factors like time and required skills indirectly translate to monetary expenses. The decisive factors for this benchmark were chosen to be wall clock time, processing costs and know-how for the generation of data cubes, and size and costs for the storage of the results. Each of them will be contextualized using the insights from chapter 2.1 and explained in detail in the following.

Wall clock time describes the *responsiveness* of the two computer systems EDC Batch Processing and mapchete Hub. It is easily obtained from mapchete Hub's REST API because the job metadata contains a *runtime* field. For simulated parallelization the *runtime* of the slowest of a set of jobs is assumed for the whole batch. EDC Batch Processing API does not expose a wall clock time parameter. Thus a work around had to be employed: The create time of the request would be compared to the last timestamp of the resulting files in the AWS S3 bucket. The difference between these points in time would be the wall clock time of a Batch Processing request.

Processing costs are—like the remaining metrics—*usage* metrics for the consumed resources. mapchete Hub consumes CPU power, S3 requests, network bandwidth and cache space. Consumed CPU resources can be easily measured by multiplying the *runtime* by the computing prices for the relevant EC2 instance type. S3 requests are made up of the *GET* requests to the satellite data repository and a small number of other requests to S3 buckets. Since the exact number of other requests cannot be determined and the *GET* requests make up at least 98 % of all S3 requests, the *GET* price is assumed for all of them. Consumed network bandwidth and cache space are very difficult to measure and do not impact the processing costs significantly. Thus they are ignored for the scope of this master thesis.

Sentinel Hub has abstracted processing costs by the so-called processing units. These are only *linear* in the case of pre-paid plans, but non-linear for the cheaper subscription plans. Both BP and mHub processing costs lack *consistency* and *independence* (the latter because they depend on either Sentinel Hub or AWS to set a price tag on processing resources).

Know-how is not a quantitative metric, but can be deduced from the technological skill set that is required to generate a data cube with the two cloud services. It is not a hard metric and doesn't offer any of the characteristics of a good performance metric. Still, human resources are a substantial part in the value chain of an IT company and must not be disregarded.

Storage size is reported because it can be a limiting factor for data portability and accessibility. Portability is deprecated in the medium term because of the paradigm shift to bring algorithms to the data instead of the other way round. Accessibility remains an issue because a huge amount of data can be difficult to post-process efficiently. The size of the computed data cubes is easy to query from the S3 bucket, however, it might be *inconsistent* between mapchete Hub and Batch Processing, depending on the chosen file format.

Storage costs are directly calculated from the storage size and are different to the other metrics in the sense that they are accounted for monthly. Thus a relevant consideration is the length of the utilization of storage services. The costs suffer from the same limitations as the storage size, and ultimately they are not even linked to the data cube generation services mapchete Hub and Batch Processing. While the standard solution to storage is to leave the generated data cubes in the destination S3 bucket, other options such as low-frequency access cloud archives or local archiving can be flexibly picked.

3.2 Test area

In order to start testing for checks by monitoring with a common test area, AMA and EOX agreed on an area of interest (AOI), which will further be referred to as MAB AOI (after the joint AMA/EOX project MAB: Monitoring Algorithm Baseline). This area was chosen with a high variety of planting zones in mind. Besides, it lies exclusively in one UTM zone (33N) and does not cross the 12° latitude to UTM zone 32N. The MAB AOI covers approximately 17,000 km² (154 km between longitudes 14.25° and 16.22° / 125 km between latitudes 47.62° and 48.75°) and contains parts of Lower Austria, Upper Austria and Styria. An overview map can be seen in Fig. 26.

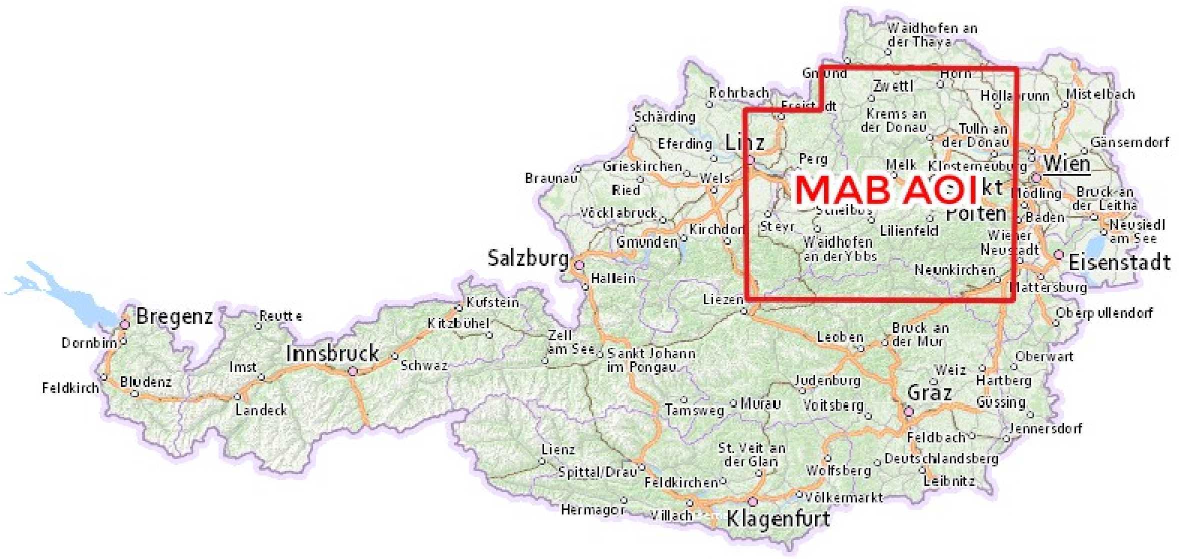


Fig. 26: Overview map of Austria showing the MAB AOI. Basemap: basemap.at. CRS: EPSG[42]:3857

Both EDC Batch Processing and mapchete Hub divide the world into tiles using a grid in order to be able to parallelize calculations. For Batch Processing—since the highest possible resolution is required in the context of this Thesis—the 100.08-km grid is not an option because its highest resolution is 60 meters. Therefore the author decided to choose the default option “s2gm”. In terms of the MAB AOI this translates to 57 tiles of 2,004 x 2,004 pixels being processed during one processing task. The relation of the tiles to the MAB AOI can be seen in Fig. 27.

[42] EPSG = CRS identifier of the European Petroleum Survey Group Geodesy

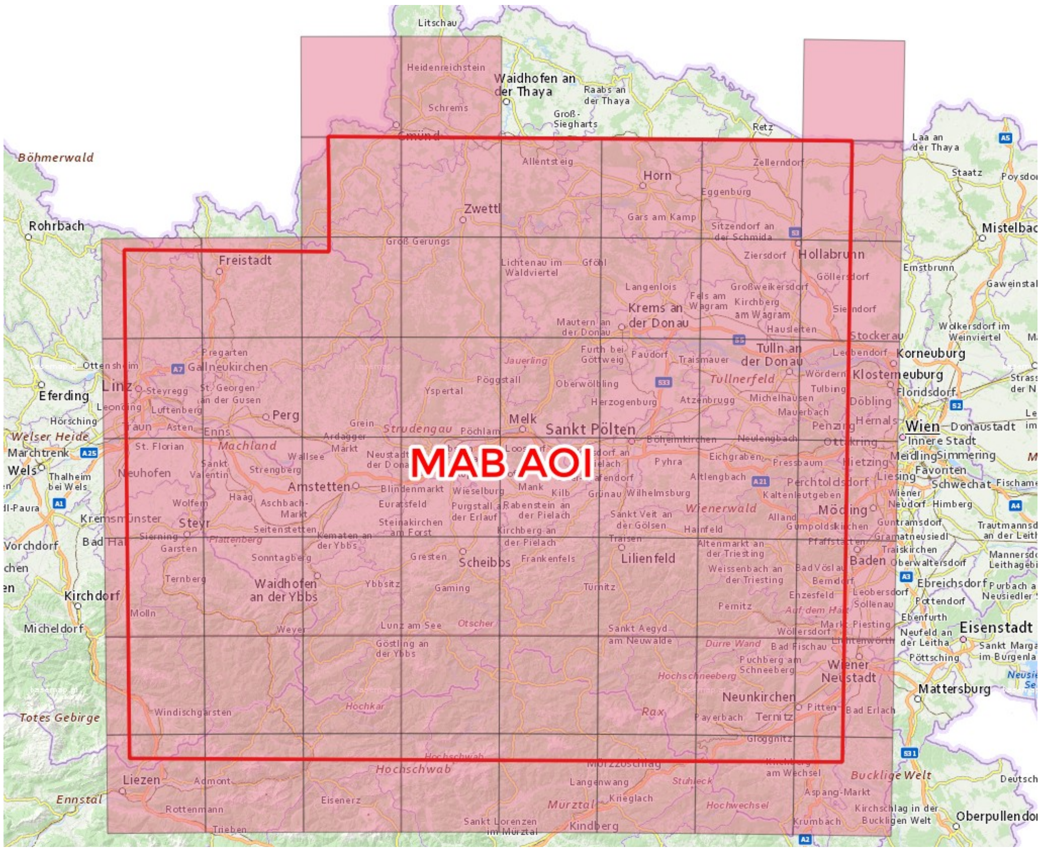


Fig. 27: The 57 S2GM tiles that make up the MAB AOI. Basemap: basemap.at. CRS: EPSG:3857

Tiling grids in mapchete Hub work very differently. They are based on a WMTS tile pyramid (see section 2.7) and pre-defined for WGS84 and Web Mercator projections. Because of their cartographic properties the pre-defined grids cannot ensure a uniform 10-m resolution for raster satellite data. Thus a custom tile pyramid had to be defined for UTM zone 33 with 1,035 tiles, each of size 81,920 x 81,920 meters (zoom level 0). The grid's extent is shown in Fig. 28.

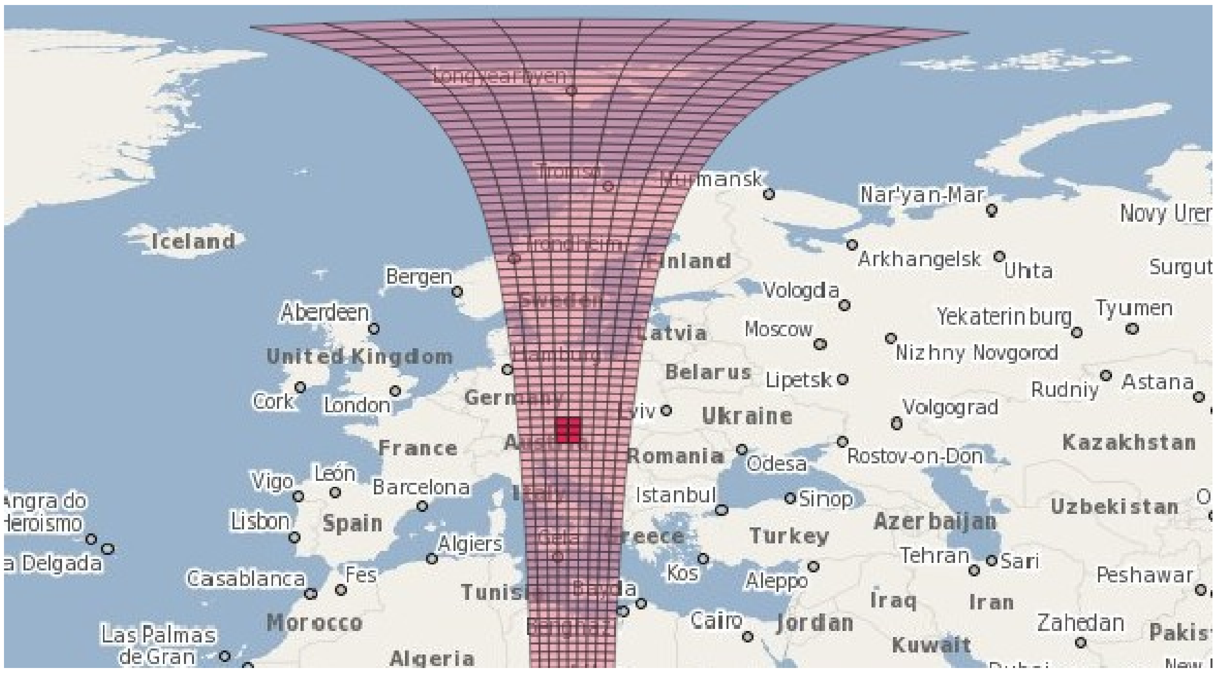


Fig. 28: Custom grid for Mapchete Hub to cover UTM zone 33 North with process zones highlighted. Basemap: OpenStreetMap (Data © OpenStreetMap contributors, Rendering © MapServer and EOX), CRS: EPSG:4326

Processing was divided into six *process zones*, with the opportunity to be parallelized during practical applications using multiple workers. In the course of this research only one worker was used to run the processes successively. The process zones are situated on zoom level 0 of the custom tile pyramid (Fig. 28) and were chosen so as to cover the Sentinel-2 tiling grid as well as possible in order to avoid too many unnecessary downloads of satellite data. Still, as Fig. 29 shows, some S2 tiles will be downloaded multiple times for more than one process zone.

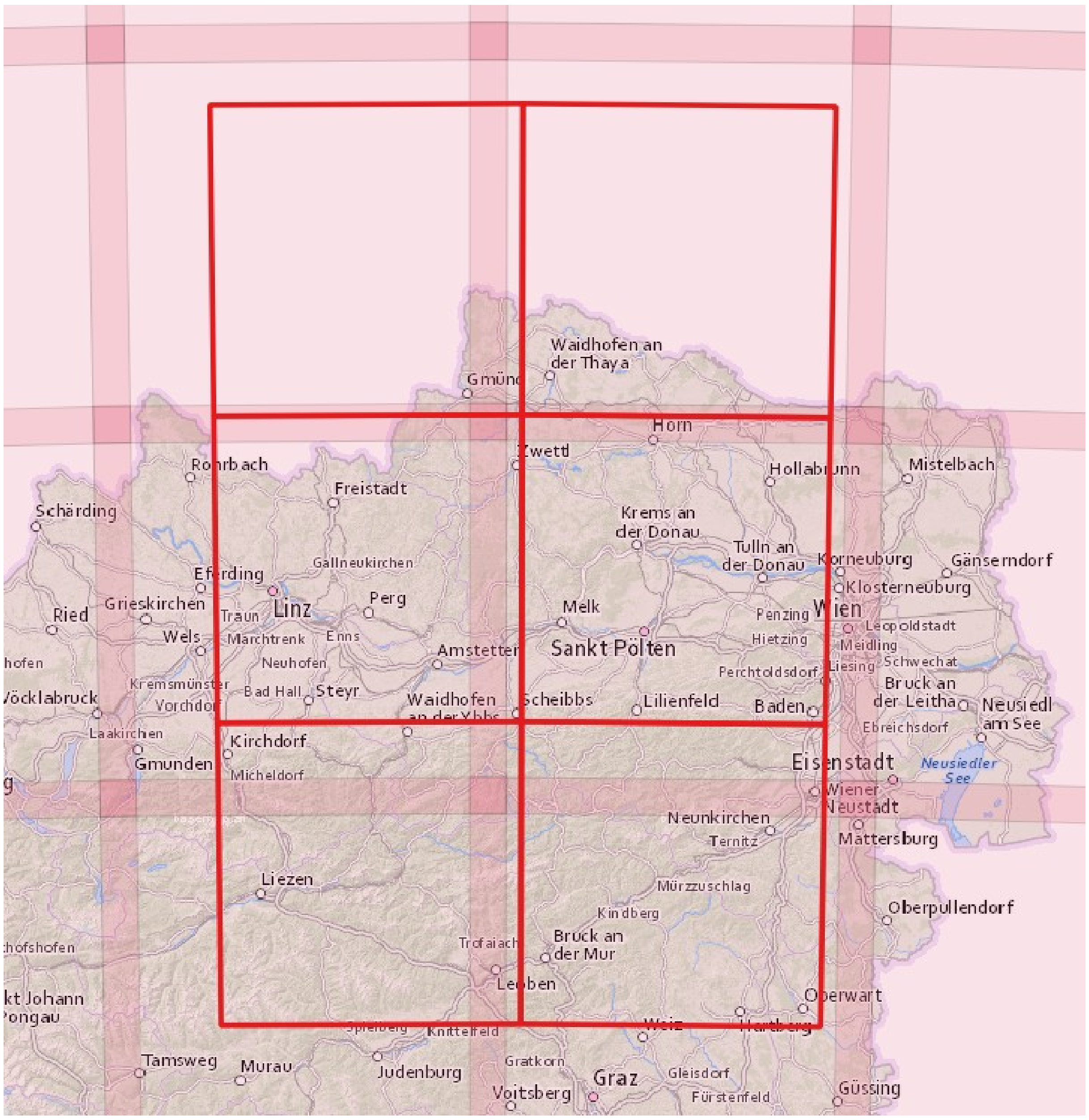


Fig. 29: Process zones (red) overlapping Sentinel-2 tiles (pink) (ESA, 2015b). Basemap: basemap.at. CRS: EPSG:3857

The actual *process tiles* use the 10-m resolution of zoom level 5, but because of metatiling activated and set to 2 they have the extent of zoom-level-4 tiles (512 x 512 pixels = 5,120 x 5,120 meters). Increasing the metatiling further did not result in performance gains. The 693 tiles intersected by the AOI are depicted in Fig. 30.

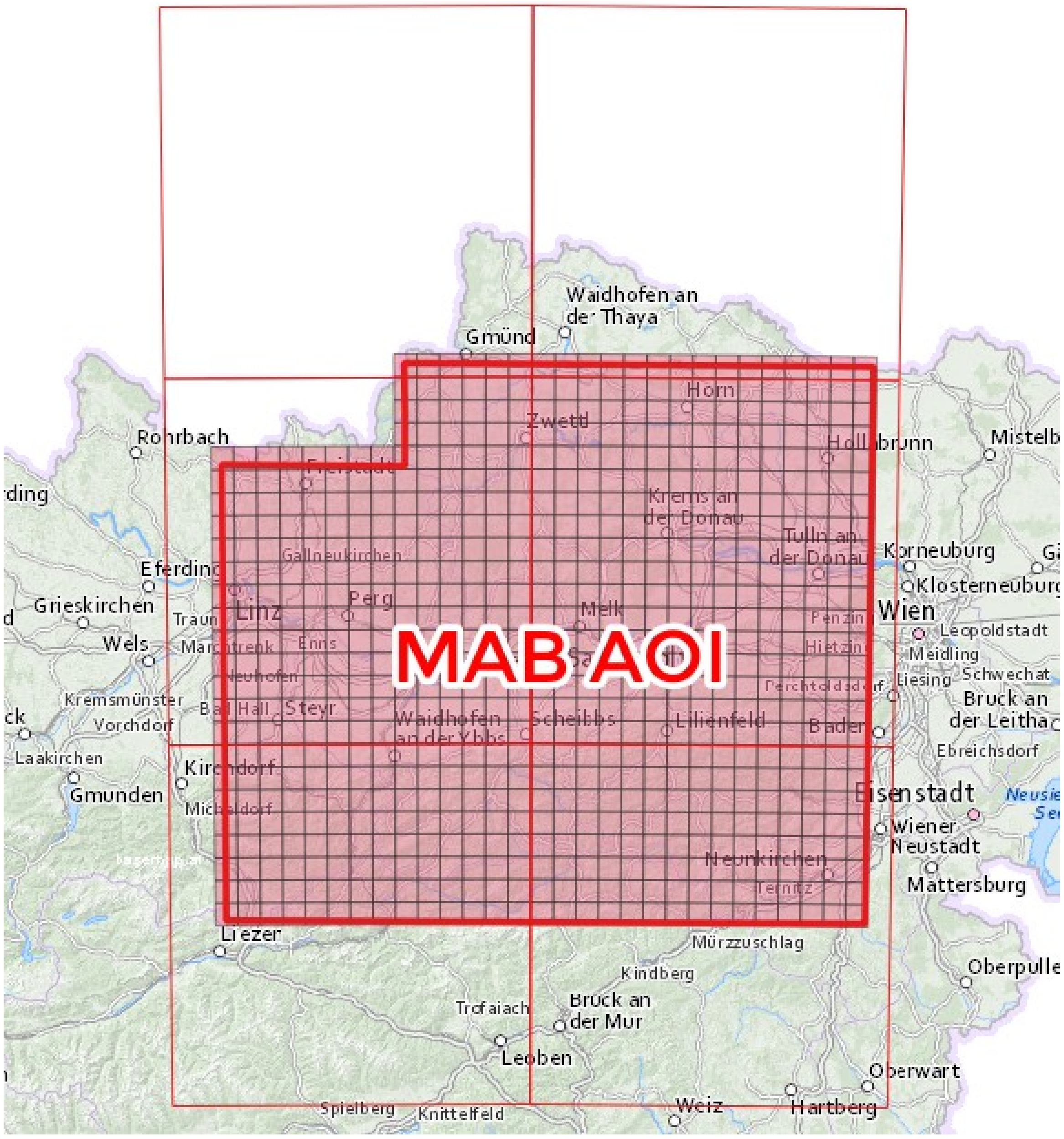


Fig. 30: Process tiles intersected by AOI in relation to process zones. Basemap: basemap.at. CRS: EPSG:3857

3.3 Data Cube Schema

The data cube schema that is followed in the course of this master's thesis is loosely modelled after the xcube Dataset Specification (the dimension names copy the xcube convention). (Brockmann Consult, 2018) The differences between the Batch Processing and mapchete Hub approaches are explained below. The respective schemas are reproduced in Table 7.

Batch Processing data cube schema	**mapchete Hub data cube schema**
Dimensions: (time: 17, x: <MAB width>, y: <MAB height>) Coordinates: * time (time) datetime64[ns] 2017-09-01–2018-05-15 * y (y) float64 <MAB height> (UTM 33N) * x (x) float64 <MAB width> (UTM 33N)	Dimensions: (time: 17, x: <MAB width>, y: <MAB height>) Coordinates: * time (time) datetime64[ns] 2019-09-01–2020-05-15 * y (y) float64 <MAB height> (UTM 33N) * x (x) float64 <MAB width> (UTM 33N)

Data variables:		Data variables:	
B02	(time, y, x) uint16 ...	B02	(time, y, x) uint16 ...
B03	(time, y, x) uint16 ...	B03	(time, y, x) uint16 ...
B04	(time, y, x) uint16 ...	B04	(time, y, x) uint16 ...
B05	(time, y, x) uint16 ...	B05	(time, y, x) uint16 ...
B06	(time, y, x) uint16 ...	B06	(time, y, x) uint16 ...
B07	(time, y, x) uint16 ...	B07	(time, y, x) uint16 ...
B08	(time, y, x) uint16 ...	B08	(time, y, x) uint16 ...
B8A	(time, y, x) uint16 ...	B8A	(time, y, x) uint16 ...
B11	(time, y, x) uint16 ...	B11	(time, y, x) uint16 ...
B12	(time, y, x) uint16 ...	B12	(time, y, x) uint16 ...
NDVI	(time, y, x) uint16 ...	NDVI	(time, y, x) uint16 ...
GNDVI	(time, y, x) uint16 ...	GNDVI	(time, y, x) uint16 ...
BNDVI	(time, y, x) uint16 ...	BNDVI	(time, y, x) uint16 ...
CVI	(time, y, x) float32 ...	CVI	(time, y, x) uint16 ...
NDSI	(time, y, x) uint16 ...	NDSI	(time, y, x) uint16 ...
NDWI	(time, y, x) uint16 ...	NDWI	(time, y, x) uint16 ...

Table 7: Comparison of data cube schemas of the data cubes generated using Batch Processing (left) and mapchete Hub (right). Differences highlighted in grey.

The target dimensions spatially cover the MAB AOI and temporally span 17 time slices. These are defined as half-month slices (1st to 15th/16th to last day of the month) that describe the averaged data of all satellite observations in that half-month. The data cube shall contain data of the 8.5 months between the start of the winter season on 1st September and the first due date on 15th May of the following year.

The variables were chosen so as to match those used in the machine learning pipeline that is developed in the MAB project. There are 16 variables packed into the data cube, of which ten are original satellite bands (variables starting in Bxx) and six are derived vegetation, water and salinity indices (IDB, 2020):

- NDVI (Normalized Difference Vegetation Index)
- GNDVI (Green Normalized Difference Vegetation Index)
- BNDVI (Blue Normalized Difference Vegetation Index)
- CVI (Chlorophyll vegetation index)
- NDSI (Normalized Difference Salinity Index)
- NDWI (Normalized Difference Water Index)

There are two differences between the BP and mHub schemas: One is the different time range and the other one concerns the data type of the CVI variable. The BP data cube contains data from September 2017 to May 2018, which is the original test time range of the MAB project. Since some S2 granules in the AWS data source that mHub connects to were missing the CRS (coordinate reference system) metadata, it was decided for the mHub data cube to use data from September 2019 to May 2020, which has correct CRS data attached to it.

The normalized-difference indices are defined in the interval [-1; 1]; thus they can easily be mapped to the interval [0; 65,535], which is the data space of a *uint16* variable (16-bit unsigned integer → 2^{16} = 65,536 non-negative values). The CVI[43] is—per definition—not constrained to the range [-1; 1], but

[43] $(\rho_{NIR}/\rho_{green}) * (\rho_{red}/\rho_{green})$

in theory goes up to infinity; a real-world dataset has a CVI of 12 at the 99th quantile. As a result it is not possible to naively map it to a certain interval. In order to store decimal places, we must therefore either use a *float* data type (32-bit *float32* takes twice the storage space as *uint16*) or stretch the data values by a multiplication factor. The latter approach was used for the mHub data cube: CVI values are multiplied by a factor of 1,000 conserving three decimal places. The highest CVI value that can be stored in a *uint16* variable using this technique is 65.5.

3.4 Data Cube Generation Code

This section presents the code that was used for generating the data cubes. Development of the SHM data cube script was done using a Jupyter notebook and therefore there are IPython commands embedded in that part of the code.

3.4.1 Utility function

To generate the half-monthly date intervals, a utility function was developed to be used for both applications in the benchmark. The utility function *date_interval_endpoints* takes as arguments a start time, an end time and the *day_of_new_interval*, which describes the start day of the second interval of each month. The output is a list of Python's *datetime* objects corresponding to the endpoints of the half-monthly intervals. E.g. the endpoints for September 2019 with *day_of_new_interval* set to 16 would be *[datetime(2019,9,1,0,0), datetime(2019,9,15,23,59,59), datetime(2019,9,16,0,0), datetime(2019,9,30,23,59,59)]*. The code is reproduced in 00.

First of all, hours, minutes and seconds, if any, are stripped from the start and end times. Then a list of all interval endpoints in between the start and end date is generated (if *day_of_new_*interval=16: the 1st, 15th, 16th and last day of each month). Start and end time are added to the list if not included yet anyways. Finally, the time of all right endpoints is set to 23:59:59 effectively making the intervals right-closed.

3.4.2 Batch Processing

3.4.2.1 evalscript

The code for generating a data cube using the BP service was executed in a Jupyter Lab environment on the Euro Data Cube and depends on an *evalscript* (see section 2.6.1) written in JavaScript. The evalscript contains the actual processing logic that is sent to the service and is presented in Annex B.1. It contains double curly brackets instead of single curly brackets because this is required if parsed with Python string formatting.

The evalscript is based on an example provided by Sentinel Hub (Milcinski, 2020a) and was extensively modified and expanded. Apart from the required *setup* and *evaluatePixel* functions there are several custom helper functions. The functions *validate*, *calculateIndex* and *interpolatedValue* are for the most part unchanged from the provided example (besides extending index calculation results to the interval [-1; 1] instead of [0; 1]). *fillResultArray* (and the main function *evaluatePixel*) had to be rewritten almost entirely in order to allow for modular customization via the Python execution environment. *evaluatePixel* additionally was changed from bi-weekly to outputting half-monthly intervals (half a month can have 13,15 or 16 days). The various functions are explained in detail in the following.

The *setup* function describes the input and output bands of the evalscript. The band arrays are generated dynamically inside the Jupyter Lab notebook (see below). Among the input bands there are always the *dataMask* and the *SCL* (Scene Classification Layer) bands that are evaluated in the *validate* function. The *validate* function takes a sample as input and returns “false” if the sample’s *dataMask* signals “no data” or if the sample was classified as cloud shadow, medium or high cloud probability, thin cirrus, snow/ice or as saturated or defective. In all other cases *validate* returns “true”.

For vegetation/water/salinity index calculation there is *calculateIndex*, which takes as arguments two numbers and returns the normalized difference of the two. Additionally it maps values from the interval [-1; 1] to the interval [0; 1] so that they eventually fit into a *uint16* that cannot hold negative values.

The *interpolatedValue* function takes an array of numbers as input and returns zero or the first entry of the array if the array is empty or only has one single entry, respectively. For arrays with more than one entry *interpolatedValue* returns the mean of the numbers in the array.

fillResultArray is used in the evalscript to populate the *results* object, which is eventually written to the output COGs. It takes as input an object that has the input bands and index identifiers as keys and arrays of satellite data samples for one date interval as values. First, *fillResultArray* loops through the input bands and populates *results* with the return value of *interpolatedValue* for that interval. Then the requested indices are looped through and calculated using the previously determined mean band values.

In the main part of the evalscript index components are defined (e.g. NDVI is calculated from the NIR and red bands), helper variables are inserted from the Jupyter Lab notebook (most notably the base structure of the *results* object) and the *evaluatePixel* function is described. As explained in section 2.6.1, *evaluatePixel* can receive several input arguments, of which we use the *samples* and the *scenes* arrays. The first command of the function determines whether the most recent observation is in the first or second half of the month. Then the function goes through the *samples* array and checks if the sample is outside the current interval. If so, the interval is passed to *fillResultArray* and a new interval is started. Else the sample is *validate*d and pushed to the current interval. Eventually *fillResultArray* is invoked for a last time and the *results* object is returned so that BP can write it to the output COGs.

3.4.2.2 Jupyter notebook

The workflow needs some preparations before the evalscript can be sent to the BP service and final commands in order to store metadata with the data cube. These preparations were performed in a Jupyter Lab notebook, which is why the code contains IPython commands. The whole notebook’s code is reproduced in Annex B.2.

First of all, the credentials for accessing BP are loaded and some libraries are imported. Since one of the data cube’s defining dimensions is the time, some special calculations needed to be done in this domain. Therefore two modules (*date* and *datetime*) of the Python core package *datetime* are included. *BackendApplicationClient* from the *oauthlib.oauth2* module and *OAuth2Session* from the *requests_oauthlib* package are needed to instantiate the connection to the BP service. Finally *boto3* establishes a connection to the AWS bucket that holds the data cube’s data and lets me file metadata there.

Cell 2 (Annex B.2.1) describes the process of fetching an authorization token to Sentinel Hub’s Batch API, as described in their documentation (Sentinel Hub, 2020a). The next cells up to the penultimate one configure the request that is sent to BP. The third cell (B.2.2) is intended for user input. Start and end date as well as bands and indices of the resulting data cube can be set and the target S3 bucket is defined. Using these input variables the relevant parameters for the processing request are calculated in cells 4 and 5 (B.2.3). The date interval endpoints are created using the utility function described in chapter 3.4.1. These endpoints are used to derive the interval midpoints that serve as metadata for the time axis of the data cube. Next the array templates for the evalscript and JSON payload are generated depending on the chosen input bands and indices in cell 3. It becomes apparent in line 4 of cell 5 that the resulting COGs' number of "bands" depends on the number of time slices instead of the actual satellite bands as usual.

The evalscript is ingested in cell 6 (B.2.4). As explained in section 3.4.2.1, the prepared variables are then replaced by the variables generated in the Jupyter notebook in the prior cells. In cell 7 the MAB AOI is defined using its GeoJSON representation. It is one of the parameters that make up the JSON payload of the processing request. Besides, we have to choose the type of satellite data that will be processed (here: Sentinel-2 Level 2A), the tiling grid that is used (see chapter 2.6.1), the spatial resolution and—apart from some other values—the time range. The latter was subsequently extended starting at 2017-09-01 in order to cautiously approach the target time range of 8.5 months.

Eventually the payload is posted to the REST endpoint of the BP service. Another request is posted immediately afterwards to the *START* endpoint to kick off the processing of the request. Then the Jupyter notebook connects to the S3 bucket, where the data cube will lie, and uploads metadata such as the labels of the bands, the request ID of that specific request and a list of date interval midpoints.

The resulting file structure of the data cub can be seen in Table 8. As mentioned above, there is one COG per satellite band or vegetation index and each COG stores the time series for that band/index. This is in contrast to conventional GeoTIFF use; since GeoTIFFs have no concept of time they would usually store the *satellite bands* as their layers.

COGs	**Band contents**
B02.tif	2017-09-01–2017-09-15 2017-09-15–2017-09-30 …
B03.tif	2017-09-01–2017-09-15 2017-09-15–2017-09-30 …
B04.tif	2017-09-01–2017-09-15 2017-09-15–2017-09-30 …
…	…

Table 8: Structure of one Batch Processing tile in the S3 bucket. Source: own work

3.4.3 mapchete Hub

3.4.3.1 Process script

The process script for mapchete Hub is an ordinary Python script that is uploaded to the server by the *mhub* command line utility. A template was provided by EOX with the basic structure up to the point where the multi-dimensional array is available for further manipulation. The whole script is reproduced in Annex C.1.

The processing script is applied to every tile in a process zone. The first condition in the mandatory *execute* function checks whether a part of the AOI was passed to the mapchete process for the currently processed tile. If no section of the AOI is present, the tile does not intersect the AOI. In this case an empty tile is returned, nothing is written to the output directory and the next tile will be worked off.

Else a four-dimensional xarray *DataArray* is created that holds the satellite data along the requested input bands, time range and geographical coordinates of that tile. The dimensions of the *DataArray* are subsequently renamed according to their semantic meaning. Since in a *DataArray* all dimensions are equally matched, the multi-dimensional array is converted to a *DataSet*, promoting the *bands* dimension. This way the syntax of index calculations becomes more readable later.

Before continuing with indexes the half-monthly time slices are calculated. The *date_interval_endpoints* function is used again to create start and end points that are then passed to a pandas *IntervalIndex*. The *DataSet* can use this index to group satellite observations by time and automatically calculate the average per half-month. Since xarray cannot store an *IntervalIndex* into a zarr, the midpoints of the intervals are retrieved and stored instead. In the end the indices are calculated, all data is typed to *uint16* and the data cube is returned as a *DataArray* again because *mapchete_xarray* cannot to date handle *DataSet*s.

3.4.3.2 .mapchete file

The process configuration file (see Annex C.2) has multiple sections defining *input*, *output*, the tile *pyramid*, etc. First of all, the path to the process file is specified. The *input* section then contains the data that will be processed. One the one hand there is the MAB AOI, which is provided to the process as a GeoJSON file hosted in the S3 bucket of this thesis. On the other hand the satellite data to be downloaded is described. S2-L2A data with cloud masks is requested from AWS within a time range starting on September 1st, 2019 and ending one up to eight and a half months later. Some connection- and computing-related parameters can be tuned too, like the *remote_timeout* (timeout in seconds for external services if they have connection issues). *xarray* is defined as the output format and the data cube is stored as zarrs at the specified S3 bucket path. COG would not be an option because the resulting four-dimensional data cube cannot be represented properly in one COG and multiple mapchete processes would be required to reproduce the file structure that Batch Processing uses.

Finally, the custom tile pyramid is specified via its CRS (*epsg*), *bounds* and *shape. zoom_levels* and *metatiling* round off the characteristics of the resulting data cube.

The command to send the *.mapchete* file to *mapchete Hub* using *mhub* is as follows:

```
mhub -h demo-m.hub.eox.at execute /mnt/datacubes.mapchete -b 431970.0
5396920.0 513890.0 5478840.0 --queue masterdatacube_queue
```

It is issued six times: Once for each process zone with its respective bounds.

3.5 Hardware

Sentinel Hub's hardware is not disclosed to the public. Therefore only the hardware that mapchete Hub runs on can be discussed here. According to EOX (2020h, 2020g) "usually" the cluster consists of "machines with 64mem, 16cpu" of the *m5a* family. These would be *m5a.4xlarge* instances. The worker available to the master thesis experiments is limited to 8 CPUs and 32 GiB and the experiments make use of the maximum CPU load allocated, which corresponds to an *m5a.2xlarge* instance. These feature AMD EPYC 7000 series processors with up to 2.5 GHz clock speed. (AWS, 2020d)

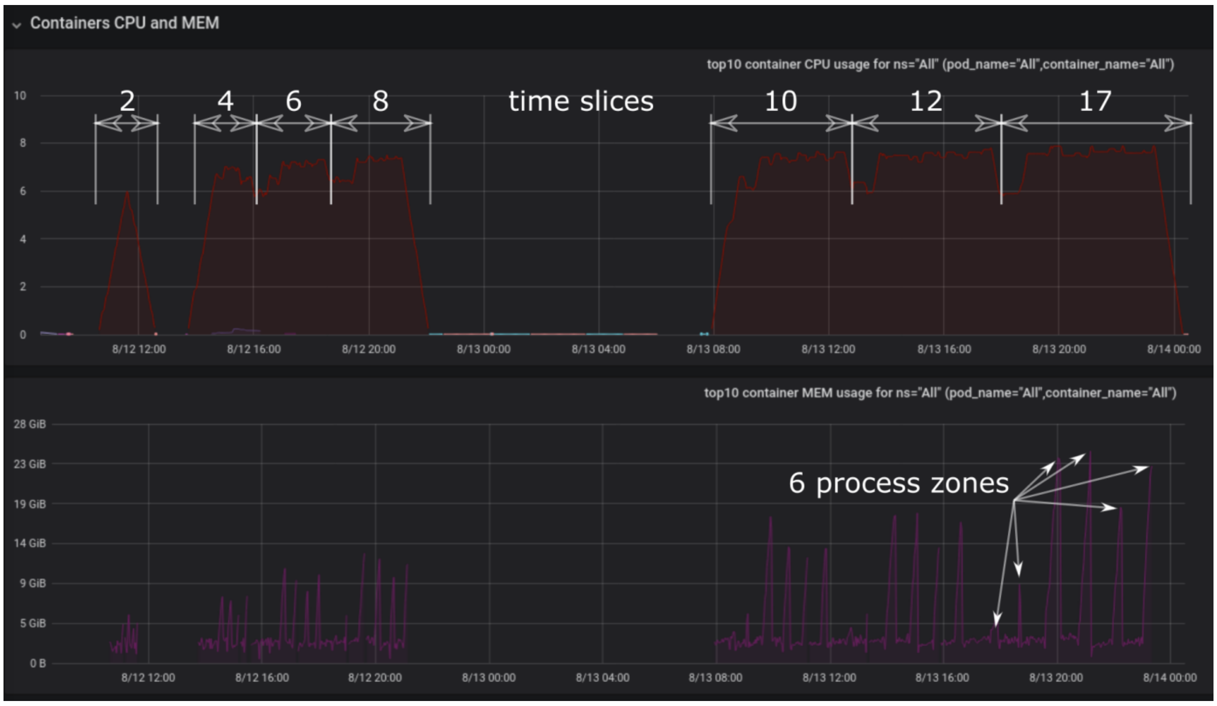


Fig. 31: Screenshot of EOX-internal CPU and memory dashboard. Annotations: own work

The experiments were conducted on August 13th and 14th, 2020 and the tracked CPU and memory loads are shown in Fig. 31. The hills in the upper diagram show the CPU load of the experiments with different time ranges. I interpret the valleys as the start of a new experiment because they only seem to occur for the upper two process zones with a very small amount of tiles to be processed (see Fig. 30). The lower diagram shows spikes for the memory consumed during each job (one job per process zone).

4 Results

This section documents the results of the experiments that were conducted during the course of the master's thesis. Performance measurements of Euro Data Cube and mapchete Hub are elaborated on and compared to each other. Furthermore, a report on the required skills to employ both tools is presented. The spatial extent of the processed data is described in section 3.2 and the measuring methods are explained in detail in chapter 3.1.

4.1 Resource usage of Euro Data Cube

Table 9 gives an overview of the measurements taken from processing the 57 s2gm tiles that cover the MAB (Monitoring Algorithm Baseline) AOI.

$n_{timeslices}$	Execution Time	Processing Costs (PU)	Size (GiB)	Storage Costs (USD/month)
2	0:08:36[a]	23 917	7.15	0.175
4	0:16:55	53 172	13.28	0.325
6	0:26:12	82 917	18.62	0.456
8	0:34:55	113 603	24.08	0.590
10	0:43:51	147 445	29.59	0.725
12	0:51:35	174 328	34.84	0.854
17	2:03:29	247 670	50.40	1.235

Table 9: Performance measurements of experiments with Sentinel Hub Batch Processing API (57 processed tiles). Grey values are inferred using linear regression. Storage costs derived from size. Prices taken from AWS website. (AWS, 2020h)

[a] **Linear regression only takes into account values for 4–12 time slices**

As can be seen in Fig. 32, execution time rose linearly for four through twelve time slices, but experienced a steep increase for 17 time slices (8 ½ months), which took over two hours to process (linear progression would be about 1:15 hours).

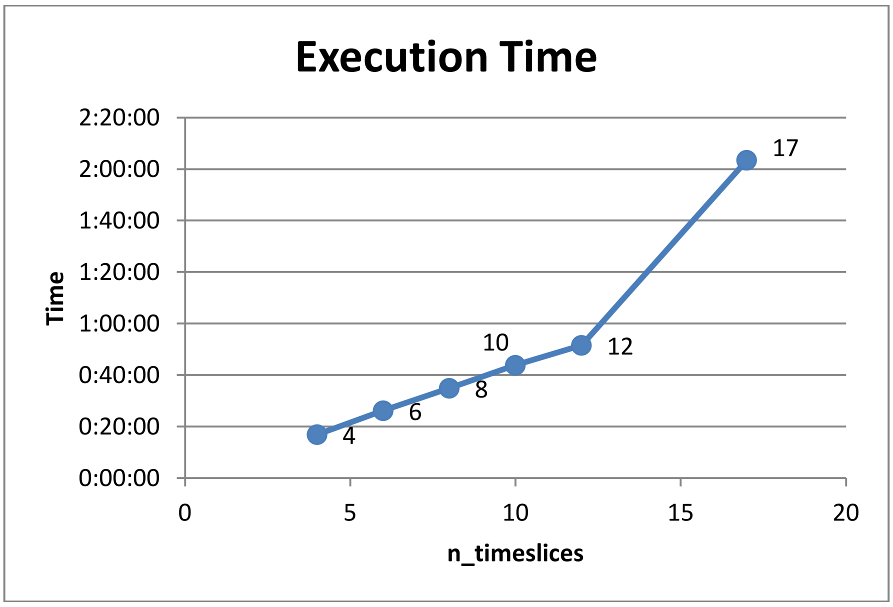


Fig. 32: Chart showing execution time plotted against the number of time slices in the data cube.

The same trend is not true for the processing costs (expressed in Processing Units). Fig. 33 demonstrates that the calculated Processing Units increased proportionally to the number of time slices in the data cube. 8 ½ months cost almost 250,000 PUs.

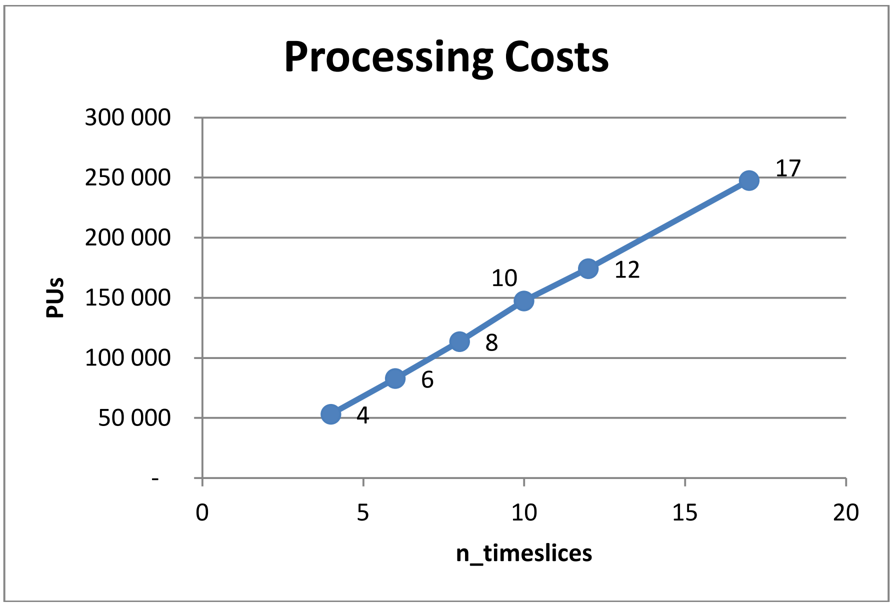


Fig. 33: Chart showing processing costs plotted against the number of time slices in the data cube.

Also the file size of the final data cube increased proportionally with the number of time slices. As Fig. 34 shows, a 17-time-slices data cube of the MAB AOI processed with Euro Data Cube occupies about 50 Gibibytes of storage space. On AWS S3 these would be USD 1.23 per month to store the data cube.

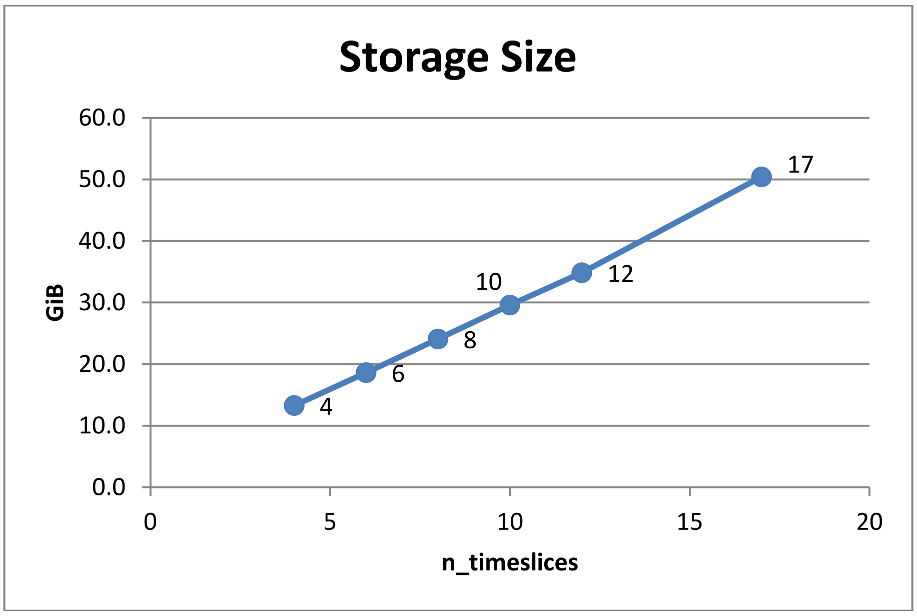


Fig. 34: Chart showing the storage size of the processed data cubes in Gibibytes plotted against the number of time slices.

As for the skill set required to generate a data cube on the Euro Data Cube, intermediate knowledge of JavaScript is a prerequisite in order to code the evalscript. Then, to send the evalscript to the EDC Batch Processing service, one must communicate with its REST API. This can either be done via a command line interface or via another script (ideally written in Python because Sentinel Hub's documentation is tailored to it). The Euro Data Cube's Jupyter Hub web interface can be used for these tasks, but they can also be completed on a local machine. Alternatively users can communicate with the API via a dedicated desktop GUI client such as Postman.

4.2 Resource usage of mapchete Hub

Results for the experiments with Mapchete Hub can be found in Table 10. Altogether 693 tiles (zoom level 5 with metatiling) were processed within 6 processing zones (zoom level 0).

$n_{timeslices}$	Execution Time	Processing Costs (USD)	Size (GiB)	Storage Costs (USD/month)
2	00:13:05	0.19	7.21	0.177
4	00:23:17	0.31	13.39	0.328
6	00:30:28	0.50	18.78	0.460
8	00:38:59	0.67	23.46	0.575
10	00:49:08	0.81	29.43	0.721
12	00:58:59	0.96	34.25	0.839
17	01:24:02	1.34	48.37	1.185

Table 10: Performance measurements of experiments with Mapchete Hub (2 756 tiles processed in 6 zones). Processing costs include costs for S3 requests and CPU time (m5a.2xlarge spot instances). Storage costs derived from size. Prices taken from AWS website. (AWS, 2020e, 2020h)

As Fig. 35 shows, the execution time rises linearly from two time slices (13 minutes) to 17 time slices (1 hour 24 minutes).

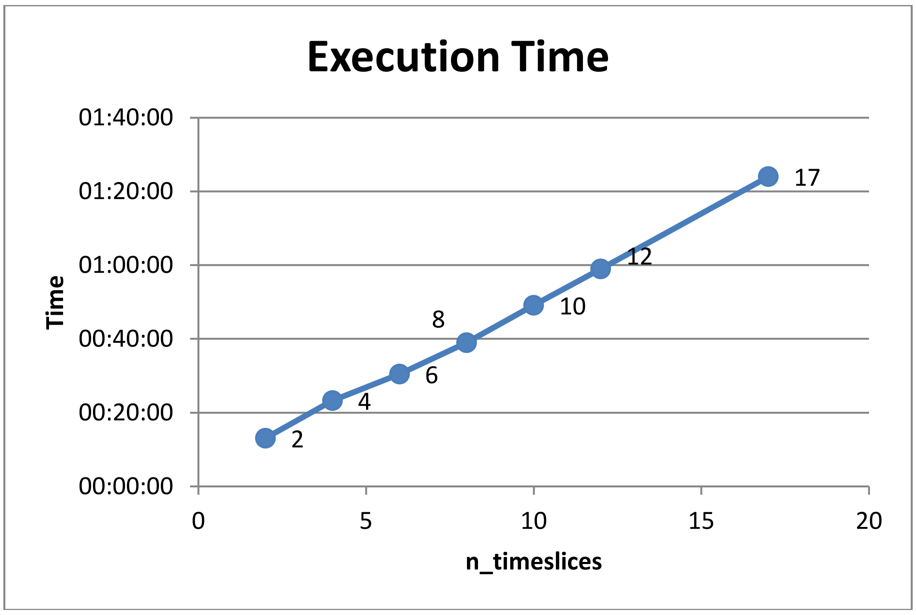


Fig. 35: Mapchete Hub: Chart showing execution time plotted against the number of time slices in the data cube.

The same observation can be made for the processing costs (see Fig. 36), albeit with a little exception: There is a bend in the results for smaller time ranges. The reasons for this bend are the S3 requests, as Fig. 37 shows more illustratively. There is a steep increase between four and eight time slices and the graph's gradient only becomes similar to the CPU time graph from 10 time slices onwards. Processing costs range between USD 0.19 for two time slices (3,708 CPU seconds and 97,357 S3 requests) and USD 1.34 for 17 time slices (24,133 / 902,624).

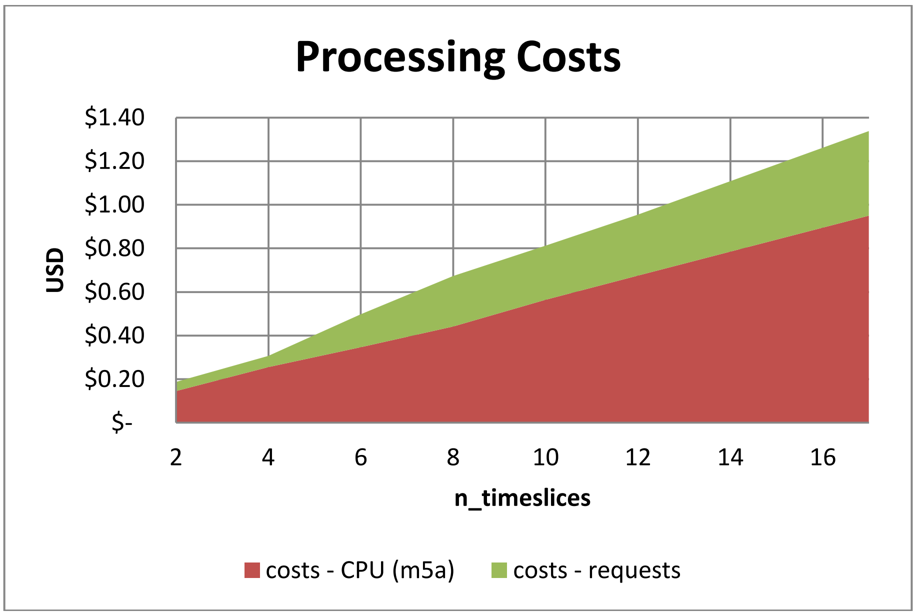


Fig. 36: Mapchete Hub: Chart showing processing costs plotted against the number of time slices in the data cube.

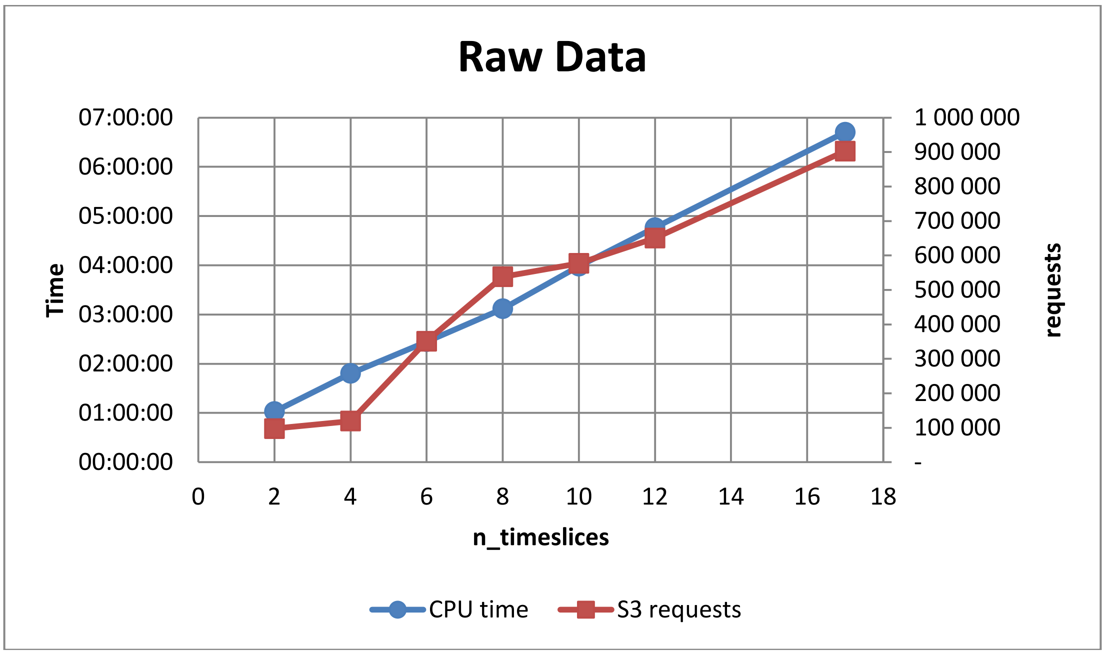


Fig. 37: Mapchete Hub: Chart showing measured CPU time and S3 requests plotted against the number of time slices in the data cube.

Storage size scales well and shows no anomalies. A two-time-slices data cube (7.21 GiB) costs USD 0.18 to store for a month in a S3 bucket, while AWS charges USD 1.19 for a 17-time-slices data cube (48.37 GiB). Details are depicted in Fig. 38.

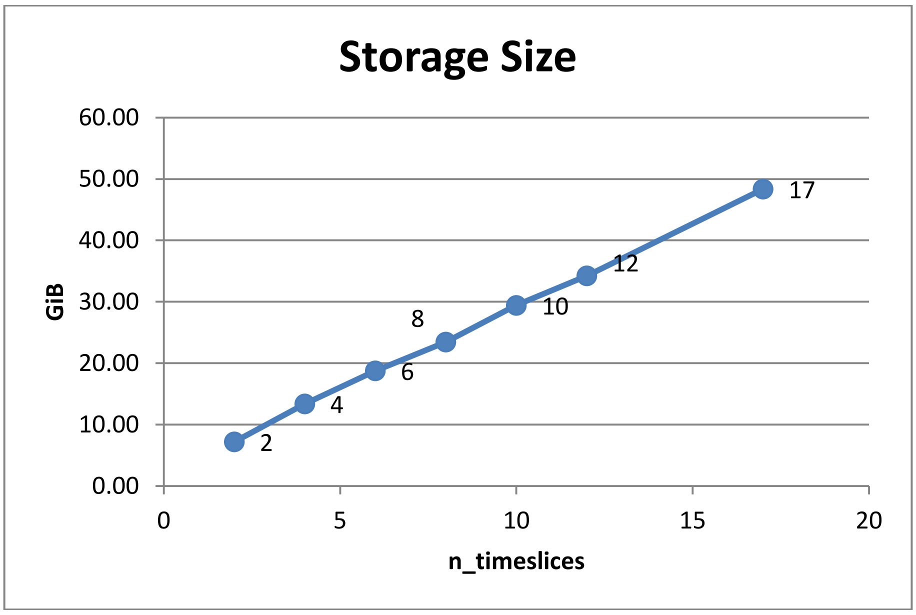


Fig. 38: Mapchete Hub: Chart showing the storage size of the processed data cubes in Gibibytes plotted against the number of time slices.

A mapchete Hub user mainly needs intermediate Python skills. Besides, for using the custom UTM grid, they must be able to define their own custom pyramid taking into account pixel sizes and zoom-levels.

4.3 Comparison

The measurements that are described in the previous sections enable a detailed comparison of *mapchete Hub* and EDC Batch Processing. Execution time and storage size of the resulting data cube can be compared directly. EDC Batch Processing performed slightly faster (4–8 minutes) than *mapchete Hub* until twelve time slices (see Fig. 39). Scaling up seemingly posed a problem to Batch Processing as execution time increased dramatically in relation to the number of processed time slices resulting in Mapchete Hub being almost 40 minutes faster for 8 ½ months. The differences in storage sizes between the COG (Batch Processing) and zarr (mapchete Hub) data cubes are very small, only amounting to 2 GiB for the largest measured data cube.

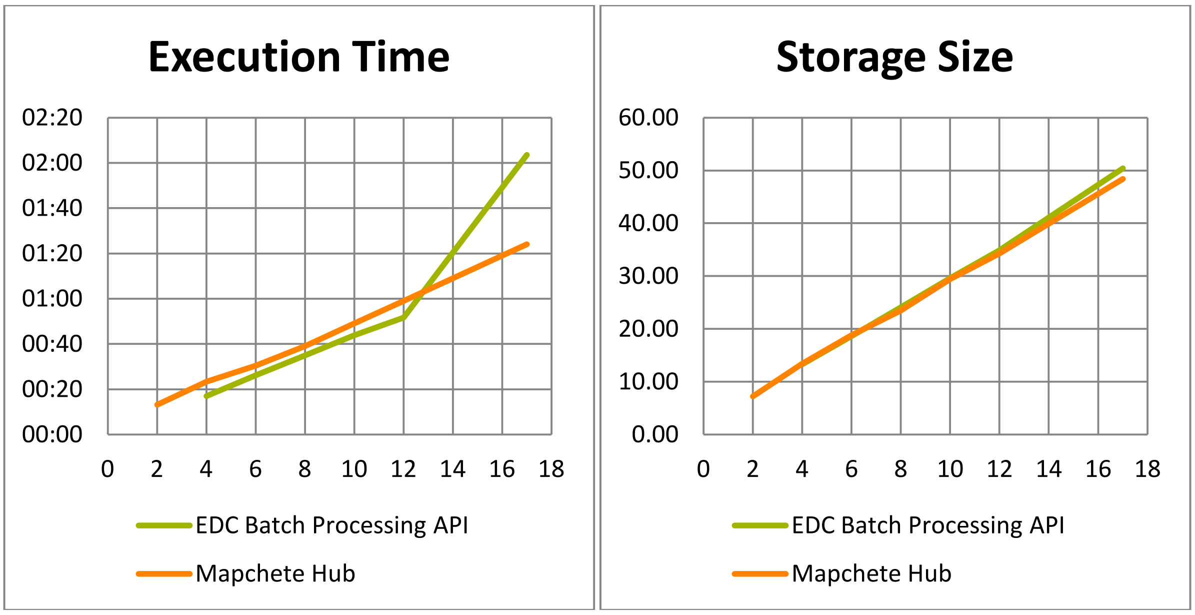


Fig. 39: Execution time (left) and storage size (right) comparison between Batch Processing and mapchete Hub

Storage costs can be deduced from the storage size and thus follow the same trend (USD 0.05 more per month for 17 time slices generated by Batch Processing). Processing costs cannot be compared easily as Batch Processing calculates with Sentinel Hub's processing units. In contrast, usage of mapchete Hub in the course of this thesis does not include any operational costs other than AWS cloud services fees. For example, while processing one month of satellite data costs USD 0.19 with mapchete Hub, it would be at least USD 24 with Batch Processing (not taking into account fees for EOx-Hub and assuming the most cost-effective subscription plan: 1,000,000 PUs for EUR 1,000). A summary of all results can be found in Table 11.

Performance indicator	**EDC Batch Processing API**	***mapchete Hub***
Execution time	Faster for ≤ six months, bad scale-up	Constant scale-up, better performance for 8 ½ months
Processing costs	Depending on subscription plan much higher[a]	Only AWS cloud services costs, very low
Storage size	Slightly bigger from eight time slices on (COG)	Slightly smaller from 8 time slices on (zarr)
Storage costs	"	"
Required skill set	Intermediate JavaScript	Intermediate Python Define custom pyramid

Table 11: Summary of performance EDC – Batch Processing API vs. Mapchete Hub
[a] Assuming the most cost-effective subscription plan: 1,000,000 PUs for EUR 1,000. Also see 2.6 and 4.4.

4.4 Scenarios

The outcome of the experiments serves as the basis for two scenarios: (1) Processing the monthly-growing data cube throughout the crop season and storing it until the end of the season, and (2) processing satellite data for the whole of Austria.

The first scenario includes generating a data cube for the 8 ½ months between September 1st and May 15th of the following year, growing the data cube monthly till November 30th. Estimating the cumulative wall clock time and processing costs at the end of the season is trivial: Throughout the season there are six additional process runs for full months (15/6, 15/7, 15/8, 15/9, 15/10, and 15/11) and on 30/11 a run for half a month. We thus assume 6.5 full-month process runs, whose results have to be added to the 8.5-months values:

Formula 2: Formula to estimate the cumulative wall clock time and processing costs to generate a data cube for the whole crop season, as well as the resulting total storage size

$$f(x) = x_{8.5} + 6.5x_1$$

Inserting into Formula 2 Batch Processing would take 2:59 and processing costs would amount to 403,000 PUs. mapchete Hub would finish processing in 2:49 cumulative wall clock time and costs would amount to USD 2.56.

Total occupied storage space is estimated in the same way using Formula 2; the data cube for the whole season will contain satellite data for 15 months. The Batch Processing data cube would have 96.87 GiB in size at the end of the season, while the mapchete Hub data cube would occupy 95.25 GiB of storage space.

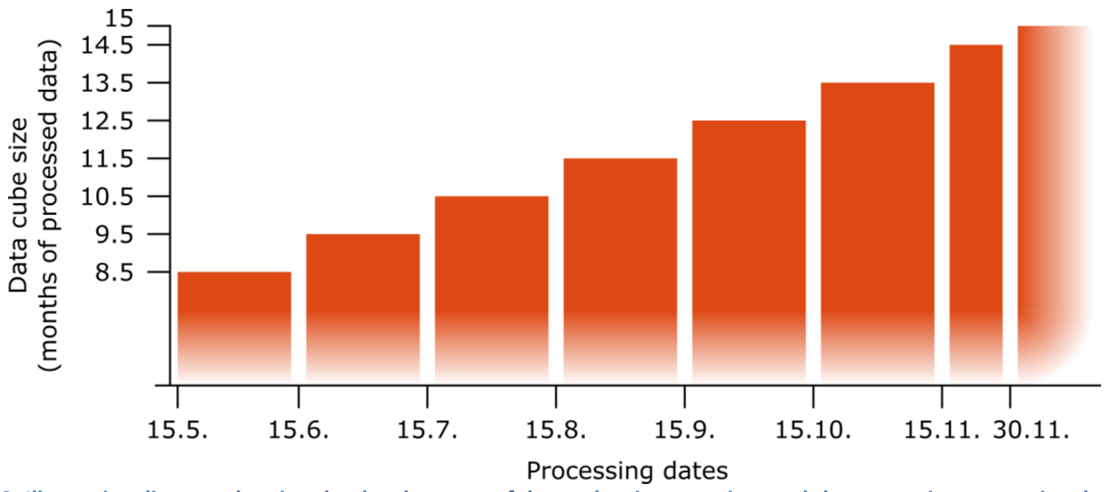


Fig. 40: Illustrative diagram showing the development of data cube size over time and the respective processing dates of the scenarios. Source: own work

Storage costs are more complicated to estimate as the storage cost bills add to each other every month. After the first month on June 15th only the costs for storing a 8.5-months data cube have to be paid; on July 15th the bill includes the initial 8.5-months data cube and two additional two half-month time slices. This goes on every 15th of the month until end of November, resulting in five (and a half) bills with additional monthly data cube slices included (an illustrative diagram can be found in Fig. 40). In order to efficiently calculate the total costs by the end of the season we must employ series calculation:

Formula 3: Storage costs calculation of scenario one.

$$storage\_costs(x) \; = 6.5x_{8.5} + \sum_{i=0}^{5}(x_1 i) + 6 * 0.5x_1$$

If we insert the values for Batch Processing into Formula 3, we get USD 11.18, while storing the map-chete Hub data cube for the whole time costs USD 10.88.

In order to estimate the outcome of scenario 2, we must first determine the number of tiles that would be processed if a data cube of Austria was generated. The EDC Batch Processing service makes it easy to find out that number. We simply create a request for the boundaries of Austria, but do not *START* it. Instead, the */tiles* endpoint is queried for a JSON representation of all the tiles of the re-quest. This JSON representation is transformed into a GeoJSON so that it can be displayed on a map and analysed (see Fig. 41). Batch Processing would in total process 266 tiles making it 4.67 times more than for the MAB AOI.

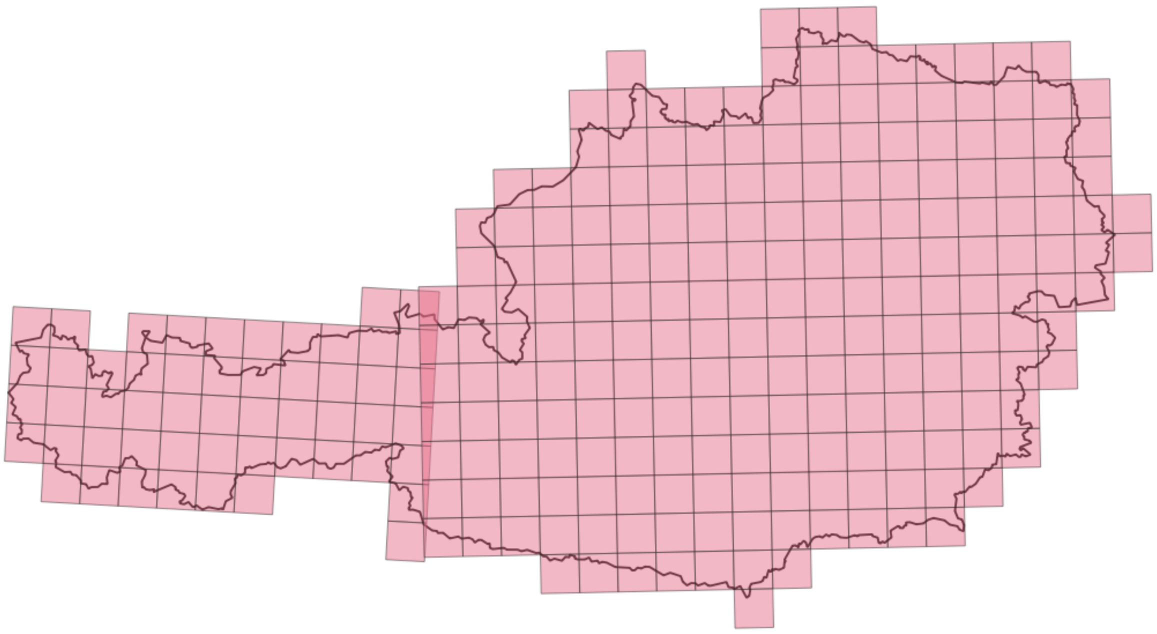


Fig. 41 All Batch Processing tiles of tile grid 0 in Austria. EPSG:31297

Doing the same investigation for mapchete Hub tiles requires more manual work. The existing custom tile pyramid for UTM zone 33N is duplicated to UTM zone 32N and the process zones that overlap completely are removed using QGIS. Then the process tiles are generated and all that lie outside Austria’s boundaries discarded. In the end 32 process zones and 3,906 process tiles remain, of which 138 tiles of UTM zone 32 overlap with 145 tiles of UTM zone 33 (see Fig. 42) amounting to an overlap of 3,404 km². Altogether 3,906 tiles are a factor of 5.64 in comparison to the MAB AOI.

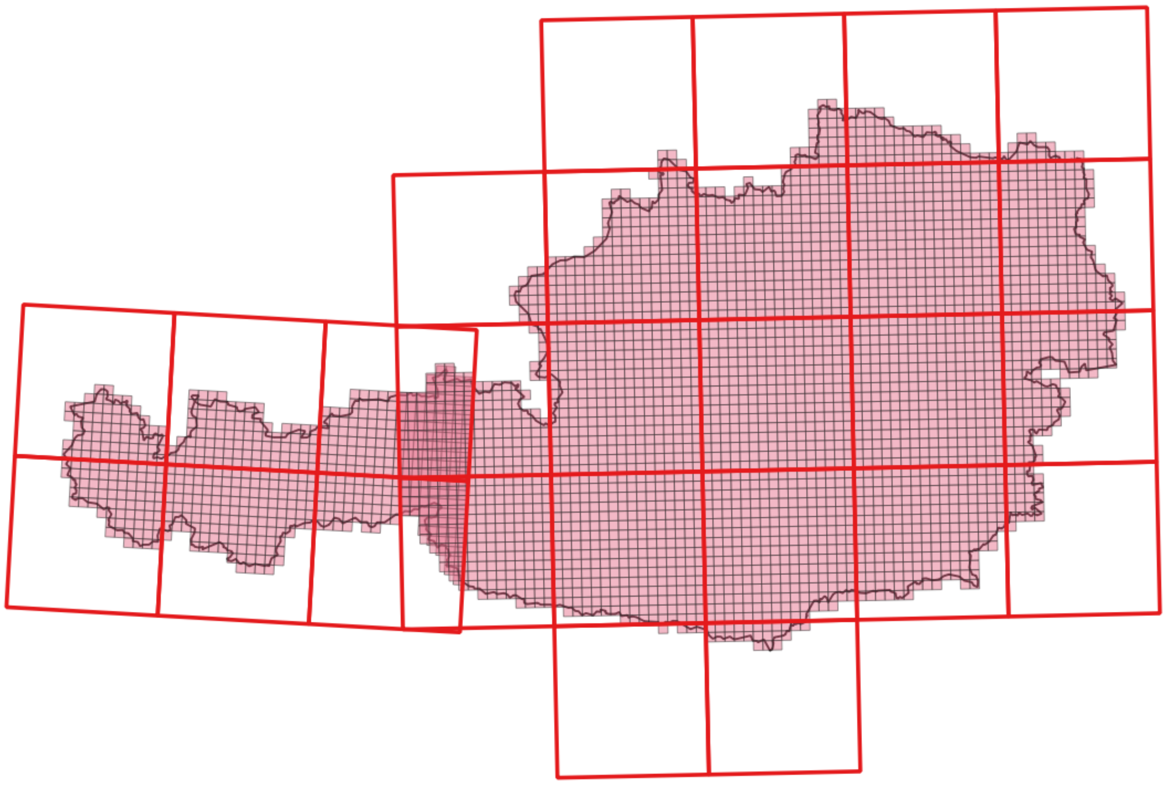


Fig. 42: All mapchete Hub process zones (red outline) and process tiles (pink) in Austria. EPSG:31297

The results of scenario 1 and 2 combined can be found in Table 12. The large number of tiles that overlap in the mapchete Hub estimate gives Batch Processing the advantage in the outcome of the benchmark. Batch Processing performs better in all categories except pricing, which arguably cannot be reliably compared.

Performance indicator	**EDC Batch Processing API**	***mapchete Hub***
Execution time	13:57	15:53
Processing costs	1 881 000 PUs	USD 14.4
Storage size	452 GiB	537 GiB
Storage costs	USD 52.2	USD 61.3

Table 12: Comparison between Batch Processing API (factor 4.67) and *mapchete Hub* (5.64) of the extrapolation of the experiment results to the whole season and the geographical extent of Austria

Still an estimate for the processing costs of Batch Processing in Euro is attempted employing the pricing schedule presented in chapter 2.6. Assuming pre-paid PUs, costs would amount to EUR 3,222 (400,000 PUs à EUR 2.5/1,000 PUs, 1,481,000 à 1.5/1,000 PUs). If subscription plans are chosen and adapted monthly we must return to the individual experiment results from chapter 4.1 and multiply by the augmentation factors to convert from the MAB AOI to Austria's extent. 1,156,619 PUs (247,670 PUs x 4.67) would result in a fee of EUR 1,235.5 (subscription for 1,000,000 PUs and remaining PUs á 1.5/1,000 PUs) for the initial data cube calculation of 8 ½ months. Additionally, monthly time slices would be at 111,692 PUs (23,917 PUs x 4.67) each. Seven subscriptions for 70,000 PUs would have to be bought and, subtracting the half month, six of them would have to be topped-up by 41,692 PUs. In all, the subscription would sum up to about EUR 2,314.

5 Discussion

In response to the main research question “Can the commercial cloud service EDC outperform the custom mHub application?” the results show that it partially can. In the experiment setup EDC has better performance in terms of wall clock time until finish of processing (except for long satellite imagery time series of more than half a year). Besides EDC users do not need the cartographical knowledge to define a custom tiling grid. However, storage space usage and resulting storage costs of the mHub data cube are lower in most cases. Processing costs cannot be compared directly, but taking into account EDC Sentinel Hub fees, mHub excels in this category. The picture is different for the extrapolation scenarios over the entire season and the whole of Austria. While processing costs of EDC Batch Processing still sky-rocket, it is about 14% faster than mHub and storage size/costs are ca. 19% lower.

This large speed and data cube size advantage of Batch Processing for a bigger area can be traced to the efficient layout of the tiling grid that Sentinel Hub has in use and the avoidance of major overlaps. This is not obvious if only one UTM zone is involved, but only becomes apparent because Austria lies in UTM zones 32 and 33.

During the experiment within the MAB AOI, Batch Processing seemed to hit a performance bottleneck for long time series of satellite data (the behaviour could be observed for eight and a half months). This could be due to memory limits that prevented Batch Processing from loading the whole time series into memory resulting in excess read and write operations to external storage. Chapter 3.5 explains that mHub had 32 GiB of memory available, which were being taken advantage of increasingly from one experiment step to the next. Nonetheless, at the maximum time series length of eight and a half months only less than 25 GiB were being made use of. So there would have been more memory resources at mHub’s disposal for further scale-up.

mapchete Hub’s execution time might also be influenced by the conversion of the cube from *DataArray* to *DataSet* and back. While Batch Processing’s evalscript is highly optimized, the processing script used for mapchete Hub converts the data for convenience. This improves the readability of the index calculations, but may have a negative impact on computing speed because of the extra conversions. On the other hand, xarray’s vectorized computations might indeed be very fast in comparison to looping through a set of *samples* in JavaScript. If nothing else, calculating the average of the satellite observation values using a pandas *IntervalIndex* is a lot less complicated than the custom evalscript functions.

The storage size of the resulting data cubes in the experiments is possibly influenced by the chosen file format and the chunking that Batch Processing and mapchete Hub apply. The first option does not offer any configuration in this regard because the four-dimensional data cube cannot be represented by COGs and therefore the “chunking” is fixed (there is one “chunk” per raster band). mapchete Hub also does not let the user specify chunking, but the zarr library uses some default values.

The enormous discrepancy in processing costs can be explained by two reasons. On the one hand mHub’s underlying software mapchete is open-source and not a commercially-sold product, while Sentinel Hub is proprietary software. On the other hand Sentinel Hub’s Processing Units include a profit margin that is not factored in for mHub.

Putting the results in relation to the extrapolated findings of Wang et al. (2019) (see chapter 0), who processed 495 GB of satellite data in 66 hours, we can note that both Batch Processing and mapchete Hub process data about 4.5 times faster than Wang et al.'s (2019) experiment setup in the OpenStack cloud. Our extrapolation, of course, did not factor in any synergetic performance improvements and simply assumed linear scaling. Therefore our extrapolation is probably biased and the difference is not as exorbitant. Nonetheless, Wang et al.'s (2019) input data and our resulting data cubes can be placed in the same league with regard to storage size.

In terms of a prospective business plan the most important factor to consider will be processing costs. In view of the whole crop season, mapchete Hub's higher wall clock time result does not constitute a relevant decision factor. Increased storage costs do play a small role, but can be mitigated by an optimization of the tiling grid. While mapchete Hub, in the context of this evaluation, needed a custom tiling pyramid, Batch Processing comes with fixed fees that easily outweigh the minor task of creating a tiling grid. Working time for developing the processing scripts incurs with both options and the surplus work of designing a tiling pyramid hardly counts in comparison to the high processing costs of Batch Processing.

Limitations of this research include the experiment design without multiple tests of the same number of time slices, which negatively impacts reliability of the results. Through linear regression the results within the same test series were counterchecked to each other. This revealed the outlier in the Batch Processing results, which can either be a systematic bottleneck or a coincidental system failure. Another limitation is the fact that networking and caching costs have been disregarded when measuring mapchete Hub's processing costs. Additionally, the differing data cube schemes theoretically increase Batch Processing's storage space and costs by 6 % (16 *uint16* bands vs. 15 *uint16* bands and one *float32* band) compared to mapchete Hub's, with the time range being another minor discrepancy. Finally, the hardware setups could not be compared because Sentinel Hub does not disclose their cloud configuration.

Despite these limitations our research contributes to the scarce records on performance of satellite data processing. Our methodology, including program code, is thoroughly disclosed and the results are presented in detail, leading to unprecedented insights into the performance of generating Earth observation data cubes.

6 Conclusions

This master thesis aimed to measure the performance of the two data cube generation tools EDC Batch Processing and mapchete Hub. The results show that in a business context Batch Processing cannot outperform mapchete Hub due to almost 200 times higher (EUR 2,314 vs. USD 14.4) processing costs for a 30-months data cube of half-monthly time slices covering Austria.

An unexpected finding is the high impact of tiling grid optimization. While EDC Batch Processing's pre-defined UTM-based tiling grid has almost no overlaps, the custom tile pyramid designed for mapchete Hub leads to duplicate processing of about 3,400 km^2 of satellite data. This in turn makes Batch Processing 14% faster than mapchete Hub and the resulting data cube 19% smaller.

Given that Batch Processing and mapchete Hub use completely different tile sizes an interesting topic for further research would be to find the optimal tile size for cloud computing. Batch Processing's performance bottleneck indicates that tile sizes (chunk sizes) cannot grow indefinitely. For the specific use case of processing long time series, optimal chunks will probably be limited in geographic extent. mapchete Hub's relatively small tile size might have led to the application handling longer time series just as well as shorter time ranges.

The same question of chunking applies to analysis performance: Some chunk sizes might yield faster analysis results than others. The data cubes that were generated during the experiments of this thesis are not optimized for analysis at all. Ideally all tiles scattered over many directories would be abstracted to one single view on the data cube. Preliminary attempts showed that xarray, in principle, can provide such a unified view over multiple files, but this feature is not implemented for COGs or zarrs (only for netCDF). Thus additional investigation is needed on how to offer a unified view on a tiled data cube in COG or zarr format.

Finally, this research only covers Sentinel-2 data, but CbM (Checks by monitoring) additionally requires Sentinel-1 radar observations. There is no need for another performance comparison between data cube services, but applying the performance metrics explored here to Sentinel-1 would be an interesting follow-up work—even if only for finding out the specifics of processing radar data.

Multiple iterations in the experiment design should be considered in future research, as reliability of the results of the present thesis is limited due to a lack of repetitions. Nevertheless this thesis contributes valuable evidence on the performance of data cube generation, which before was a largely unexplored topic.

7 Literature

AMA. (2020). *AMA Merkblatt—Allgemeine Informationen*. https://www.ama.at/getattachment/aba3cbb9-0039-4cae-a540-06dad9df167e/Merkblatt_2020_ALLGEMEIN.pdf

Augustin, H., Sudmanns, M., Tiede, D., Lang, S., & Baraldi, A. (2019). Semantic Earth Observation Data Cubes. *Data*, *4*(3), 102. https://doi.org/10.3390/data4030102

AWS. (2020a). *Amazon EC2 Dedicated Host Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/dedicated-hosts/pricing/

AWS. (2020b). *Amazon EC2 Features*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/features/

AWS. (2020c). *Amazon EC2 Instance Types*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/instance-types/

AWS. (2020d). *Amazon EC2 M5 Instances—General purpose compute workloads*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/instance-types/m5/

AWS. (2020e). *Amazon EC2 Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/pricing/

AWS. (2020f). *Amazon EC2 Spot Instances Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/spot/pricing/

AWS. (2020g). *Amazon S3*. Amazon Web Services, Inc. https://aws.amazon.com/de/s3/

AWS. (2020h). *Amazon S3 Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/s3/pricing/

AWS. (2020i). *Amazon S3 Storage Classes*. Amazon Web Services, Inc. https://aws.amazon.com/s3/storage-classes/

AWS. (2020j). *AWS Global Cloud Infrastructure*. Amazon Web Services, Inc. https://infrastructure.aws/

AWS. (2020k). *Cloud Data Archiving | Long-term Object Storage | Amazon S3 Glacier*. Amazon Web Services, Inc. https://aws.amazon.com/glacier/

AWS. (2020l). *EC2 On-Demand Instance Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/pricing/on-demand/

AWS. (2020m). *EC2 Reserved Instance Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/

AWS. (2020n). *Global Infrastructure*. Amazon Web Services, Inc. https://aws.amazon.com/about-aws/global-infrastructure/

AWS. (2020o). *Global Infrastructure Regions & AZs*. Amazon Web Services, Inc. https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

AWS. (2020p). *Savings Plans Pricing*. Amazon Web Services, Inc. https://aws.amazon.com/savingsplans/pricing/

AWS. (2020q). *Sentinel-2—Registry of Open Data on AWS*. https://registry.opendata.aws/sentinel-2/

Bala, R., Gill, B., Smith, D., & Wright, D. (2019). *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide*. Gartner. https://www.gartner.com/en/documents/3947472/magic-quadrant-for-cloud-infrastructure-as-a-service-wor

Baumann, P. (2017a). *The Datacube Manifesto*. https://external.opengeospatial.org/twiki_public/pub/CoveragesDWG/Datacubes/The-Datacube-Manifesto.pdf

Baumann, P. (2017b). Standardizing big earth datacubes. *2017 IEEE International Conference on Big Data (Big Data)*, 67–73. https://doi.org/10.1109/BigData.2017.8257912

Brockmann Consult. (2018, May 31). *Xcube Dataset Specification*. https://xcube.readthedocs.io/en/latest/cubespec.html#basic-schema

Celery. (2020a). *First Steps with Celery—Celery 4.4.7 documentation*. https://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html

Celery. (2020b, May 16). *Homepage | Celery: Distributed Task Queue*. https://web.archive.org/web/20200516055757/http://www.celeryproject.org/

Chuvieco, E. (2020). *Fundamentals of Satellite Remote Sensing: An Environmental Approach, Third Edition*. CRC Press LLC. http://ebookcentral.proquest.com/lib/univie/detail.action?docID=6026405

cogeotiff. (2020). *Cloud Optimized GeoTIFF in depth*. https://www.cogeo.org/in-depth.html

Devos, W., Fasbender, D., Griffiths, P., Lemoine, G., Loudjani, P., Milenov, P., Sima, A., Wirnhardt, C., European Commission, & Joint Research Centre. (2017). *Second discussion document on the introduction of monitoring to substitute OTSC: Rules for processing applications in 2018-2019.* http://publications.europa.eu/publication/manifestation_identifier/PUB_KJNA29369ENN

Devos, W., Fasbender, D., Lemoine, G., Milenov, P., European Commission, & Joint Research Centre. (2017). *Technical guidance on the decision to go for substitution of OTSC by monitoring.* http://publications.europa.eu/publication/manifestation_identifier/PUB_KJNA29370ENN

Du, Y., Zhang, Y., Ling, F., Wang, Q., Li, W., & Li, X. (2016). Water Bodies' Mapping from Sentinel-2 Imagery with Modified Normalized Difference Water Index at 10-m Spatial Resolution Produced by Sharpening the SWIR Band. *Remote Sensing*, *8*(4), 354. https://doi.org/10.3390/rs8040354

Eamus, D., Huete, A., & Yu, Q. (2016). *Vegetation Dynamics: A Synthesis of Plant Ecophysiology, Remote Sensing and Modelling*. Cambridge University Press. https://doi.org/10.1017/CBO9781107286221

EDC Consortium. (2020a). *Euro Data Cube*. https://eurodatacube.com/

EDC Consortium. (2020b). *Euro Data Cube Services*. https://cockpit.hub.eox.at/storage/uploads/edc-editor/Euro_Data_Cube_summary_brochure.pdf

EOX. (2020a). *Mapchete Hub*. https://gitlab.eox.at/maps/mapchete_hub

EOX. (2020b). *Mapchete Process Configuration—Mapchete 0.35 documentation*. https://mapchete.readthedocs.io/en/stable/configuration.html

EOX. (2020c). *Mapchete Process—Mapchete 0.35 documentation*. https://mapchete.readthedocs.io/en/stable/processes.html

EOX. (2020d). *Mapchete Satellite*. https://gitlab.eox.at/maps/mapchete_satellite

EOX. (2020e). *Output Formats—Mapchete 0.35 documentation*. https://mapchete.readthedocs.io/en/stable/process_output.html

EOX. (2020f). *Tiling and projections—Mapchete 0.35 documentation*. https://mapchete.readthedocs.io/en/latest/tiling.html

EOX. (2020). *Ungarj/mapchete_xarray* [Python]. https://github.com/ungarj/mapchete_xarray (Original work published 2019)

EOX. (2020). *Ungarj/mapchete* [Python]. https://github.com/ungarj/mapchete (Original work published 2015)

EOX. (2020g, August 13). *Think we had this discussion already...* https://eox.slack.com/archives/CU590JHB9/p1597300810008900?thread_ts=1597244595.004700&cid=CU590JHB9

EOX. (2020h, August 13). *Usually in our cluster...* https://eox.slack.com/archives/CU590JHB9/p1597301563009500?thread_ts=1597244595.004700&cid=CU590JHB9

ESA. (2015a). *Bulletin 161*.

ESA. (2015b). *Sentinel-2 tiling grid* [Map]. https://sentinel.esa.int/documents/247904/1955685/S2A_OPER_GIP_TILPAR_MPC__20151209T095117_V20150622T000000_21000101T000000_B00.kml

ESA. (2018a, May 25). *Sentinels modernise Europe's agricultural policy*. http://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinels_modernise_Europe_s_agricultural_policy

ESA. (2018b). *STATEMENT OF WORK - Procurement of EO Data Cube Facility Service: 2018-2022*.

ESA. (2020a). *Copernicus Open Access Hub*. https://scihub.copernicus.eu/

ESA. (2020b). *Level-2A Algorithm*. Sentinel-2 MSI Technical Guide - Sentinel Online. https://sentinels.copernicus.eu/web/sentinel/technical-guides/sentinel-2-msi/level-2a/algorithm

ESA. (2020c). *Overview*. http://www.esa.int/Applications/Observing_the_Earth/Copernicus/Overview4

ESA. (2020d). *Product Types*. Sentinel-2 MSI - User Guides - Sentinel Online. https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/product-types

ESA. (2020e). *Radiometric Resolutions*. Sentinel-2 MSI - User Guides - Sentinel Online. https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions/radiometric

ESA. (2020f). *Revisit and Coverage*. Sentinel-2 MSI - User Guides - Sentinel Online. https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/revisit-coverage

EU. (2018a). *Commission Implementing Regulation (EU) 2018/746 of 18 May 2018 amending Implementing Regulation (EU) No 809/2014 as regards modification of single applications and payment claims and checks*. http://data.europa.eu/eli/reg_impl/2018/746/oj/eng

EU. (2018b, May 25). *Modernising the CAP: Satellite data authorised to replace on-farm checks* [Text]. European Commission - European Commission. https://ec.europa.eu/info/news/modernising-cap-satellite-data-authorised-replace-farm-checks-2018-may-25_en

EU. (2020a). *Copernicus In Brief*. https://www.copernicus.eu/en/about-copernicus/copernicus-brief

EU. (2020b). *Integrated Administration and Control System (IACS)* [Text]. European Commission - European Commission. https://ec.europa.eu/info/food-farming-fisheries/key-policies/common-agricultural-policy/financing-cap/financial-assurance/managing-payments_en

EU. (2020c). *The common agricultural policy at a glance* [Text]. European Commission - European Commission. https://ec.europa.eu/info/food-farming-fisheries/key-policies/common-agricultural-policy/cap-glance_en

European Commission. (n.d.). *Infrastructure | Copernicus*. Retrieved April 15, 2020, from https://www.copernicus.eu/en/about-copernicus/infrastructure

Garner, B. (2020, April 24). *Celery Tutorial: A Must-Learn Technology for Python Developers*. Medium. https://medium.com/swlh/python-developers-celery-is-a-must-learn-technology-heres-how-to-get-started-578f5d63fab3

Giuliani, G., Camara, G., Killough, B., & Minchin, S. (2019). Earth Observation Open Science: Enhancing Reproducible Science Using Data Cubes. *Data*, *4*(4), 147. https://doi.org/10.3390/data4040147

Giuliani, G., Chatenoux, B., De Bono, A., Rodila, D., Richard, J.-P., Allenbach, K., Dao, H., & Peduzzi, P. (2017). Building an Earth Observations Data Cube: Lessons learned from the Swiss Data Cube (SDC) on generating Analysis Ready Data (ARD). *Big Earth Data*, *1*(1–2), 100–117. https://doi.org/10.1080/20964471.2017.1398903

Giuliani, G., Chatenoux, B., Honeck, E., & Richard, J.-P. (2018). Towards Sentinel-2 Analysis Ready Data: A Swiss Data Cube Perspective. *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, 8659–8662. https://doi.org/10.1109/IGARSS.2018.8517954

Giuliani, G., Masó, J., Mazzetti, P., Nativi, S., & Zabala, A. (2019). Paving the Way to Increased Interoperability of Earth Observations Data Cubes. *Data*, *4*(3), 113. https://doi.org/10.3390/data4030113

Gomes, V. C. F., Queiroz, G. R., & Ferreira, K. R. (2020). An Overview of Platforms for Big Earth Observation Data Management and Analysis. *Remote Sensing*, *12*(8), 1253. https://doi.org/10.3390/rs12081253

Han, J., Kamber, M., & Pei, J. (2012). Data Cube Technology. In *Data Mining* (pp. 187–242). Elsevier. https://doi.org/10.1016/B978-0-12-381479-1.00005-8

HDF Group. (2017). *HDF5 Sell Sheet*. https://www.hdfgroup.org/wp-content/uploads/2017/12/HDF512-17.pdf

IDB. (2020). *IDB - List of available Indices*. https://www.indexdatabase.de/db/i.php

King, M., & Herring, D. (2001). Research Satellites for Atmospheric Science, 1978-Present. In J. Holton, J. Pyle, & J. Curry (Eds.), *Encyclopedia of Atmospheric Sciences*. Academic Press. https://earthobservatory.nasa.gov/features/RemoteSensingAtmosphere

Knorr, E. (2020). The 2020 IDG Cloud Computing Survey. *InfoWorld.Com*.

Knowelden, R., & Castriotta, A. G. (2020). *Copernicus Sentinel Data Access 2019—Annual Report*. https://scihub.copernicus.eu/twiki/pub/SciHubWebPortal/AnnualReport2019/COPE-SERCO-RP-20-0570_-_Sentinel_Data_Access_Annual_Report_Y2019_v1.0.pdf

Koetz, Defourny, Bontemps, Bajec, Cara, de Vendictis, Kucera, Malcorps, Milcinski, Nicola, Rossi, Sciarretta, Slacikova, Tutunaru, Udroiu, & Zavagl. (2019). *SEN4CAP - Sentinels for CAP monitoring approach*.

Kopp, S., Becker, P., Doshi, A., Wright, D. J., Zhang, K., & Xu, H. (2019). Achieving the Full Vision of Earth Observation Data Cubes. *Data*, *4*(3), 94. https://doi.org/10.3390/data4030094

Lewis, A., Lymburner, L., Purss, M. B. J., Brooke, B., Evans, B., Ip, A., Dekker, A. G., Irons, J. R., Minchin, S., Mueller, N., Oliver, S., Roberts, D., Ryan, B., Thankappan, M., Woodcock, R., & Wyborn, L. (2016). Rapid, high-resolution detection of environmental change over continental scales from satellite data – the Earth Observation Data Cube. *International Journal of Digital Earth*, *9*(1), 106–111. https://doi.org/10.1080/17538947.2015.1111952

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., … Wang, L.-W. (2017). The Australian Geoscience Data Cube—Foundations and lessons learned. *Remote Sensing of Environment*, *202*, 276–292. https://doi.org/10.1016/j.rse.2017.03.015

Lilja, D. J. (Ed.). (2000). Metrics of performance. In *Measuring Computer Performance: A Practitioner's Guide* (pp. 9–24). Cambridge University Press; Cambridge Core. https://doi.org/10.1017/CBO9780511612398.003

Loudjani, P. (2019). *… After early years and those to come …*. https://ec.europa.eu/jrc/sites/jrcsh/files/1-tuesday26-11.zip

Masó, J. (2016, January 19). *OGC® Web Map Tile Service (WMTS) Simple Profile* [Implementation Standard]. WMS SWG; Open Geospatial Consortium. http://docs.opengeospatial.org/is/13-082r2/13-082r2.html

McFeeters, S. K. (1996). The use of the Normalized Difference Water Index (NDWI) in the delineation of open water features. *International Journal of Remote Sensing*, *17*(7), 1425–1432. https://doi.org/10.1080/01431169608948714

Mell, P. M., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. https://www.nist.gov/publications/nist-definition-cloud-computing

Mesnier, M., Ganger, G. R., & Riedel, E. (2003). Object-based storage. *IEEE Communications Magazine*, *41*(8), 84–90. https://doi.org/10.1109/MCOM.2003.1222722

Milcinski, G. (2020a). *Eo-learn_Batch.postman_collection.json* [JSON].

Milcinski, G. (2020b, January 7). *Large-scale data preparation—Introducing Batch Processing*. Medium. https://medium.com/sentinel-hub/large-scale-data-preparation-introducing-batch-processing-b3a58755b8a1

Milcinski, G. (2020c, June 16). *Euro Data Cube*. https://www.youtube.com/watch?time_continue=108&v=lqckgCfucas&feature=emb_logo

MongoDB. (2020a). *Document Database*. MongoDB. https://www.mongodb.com/document-databases

MongoDB. (2020b). *The most popular database for modern apps*. MongoDB. https://www.mongodb.com

Moreira, A., Prats-Iraola, P., Younis, M., Krieger, G., Hajnsek, I., & Papathanassiou, K. P. (2013). A tutorial on synthetic aperture radar. *IEEE Geoscience and Remote Sensing Magazine*, *1*(1), 6–43. https://doi.org/10.1109/MGRS.2013.2248301

Murugesan, S., & Bojanova, I. (2016). Cloud Computing. In *Encyclopedia of Cloud Computing* (pp. 1–14). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118821930.ch1

NASA. (2020, August 24). *The Thematic Mapper*. Landsat Science. https://landsat.gsfc.nasa.gov/the-thematic-mapper/

Nativi, S., Mazzetti, P., & Craglia, M. (2017). A view-based model of data-cube to support big earth data systems interoperability. *Big Earth Data*, *1*(1–2), 75–99. https://doi.org/10.1080/20964471.2017.1404232

Obaidat, M., & Boudriga, N. (2010). Introduction and Basic Concepts. In *Fundamentals of Performance Evaluation of Computer and Telecommunication Systems* (pp. 1–20). John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470567203.ch1

OGC. (2010). *OGC® WCS 2.0 Interface Standard* (P. Baumann, Ed.).

Pettorelli, N., Schulte to Buehne, H., Shapiro, A., & Glover-Kapfer, P. (2018). *Conservation Technology Series Issue 4: SATELLITE REMOTE SENSING FOR CONSERVATION*. https://doi.org/10.13140/RG.2.2.25962.41926

Purss, M. B. J., Peterson, P. R., Strobl, P., Dow, C., Sabeur, Z. A., Gibb, R. G., & Ben, J. (2019). Datacubes: A Discrete Global Grid Systems Perspective. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *54*(1), 63–71.

Sentinel Hub. (2020a). *Authentication*. https://docs.sentinel-hub.com/api/latest/#/API/authentication?id=python

Sentinel Hub. (2020b). *Batch Processing API*. https://docs.sentinel-hub.com/api/latest/#/BATCH_API/batch_processor?id=tiling-grids

Sentinel Hub. (2020c). *Definition of a Processing Unit*. https://docs.sentinel-hub.com/api/latest/api/overview/processing-unit/

Sentinel Hub. (2020d). *Evalscript v3*. https://docs.sentinel-hub.com/api/latest/evalscript/v3/

Sentinel Hub. (2020e). *Examples of Batch Processing Workflow*. https://docs.sentinel-hub.com/api/latest/api/batch/examples/

SEOS. (n.d.). *Introduction to Remote Sensing*. Retrieved April 9, 2020, from https://seos-project.eu/remotesensing/remotesensing-c01-p06.html

Siegmund, A., & Menz, G. (2005). Fernes nah gebracht—Satelliten- und Luftbildeinsatz zur Analyse von Umweltveränderungen im Geographieunterricht. In *Geographie und Schule* (Vol. 27, Issue 154, pp. 2–10, S. 49).

Strobl, P., Baumann, P., Lewis, A., Szantoi, Z., Killough, B., Purss, M., Craglia, M., Nativi, S., Held, A., & Dhu, T. (2017). *The six faces of the data cube*. https://doi.org/10.2760/383579

Sudmanns, M., Tiede, D., Lang, S., Bergstedt, H., Trost, G., Augustin, H., Baraldi, A., & Blaschke, T. (2020). Big Earth data: Disruptive changes in Earth observation data management and analysis? *International Journal of Digital Earth*, *13*(7), 832–850. https://doi.org/10.1080/17538947.2019.1585976 the pandas development team. (2020). *Intro to data structures*. Pandas 1.1.1 Documentation. https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe

The SciPy community. (2020). *NumPy: The absolute basics for beginners*. NumPy v1.20.Dev0 Manual. https://numpy.org/devdocs/user/absolute_beginners.html

Ungar, J. (2020a). *Datacubes.mapchete* [Python]. EOX. https://github.com/StefanBrand/masterdatacube

Ungar, J. (2020b). *Xarray_export.py* [Python]. EOX. https://github.com/StefanBrand/masterdatacube

Unidata. (2018). *NetCDF Factsheet*. https://www.unidata.ucar.edu/publications/factsheets/current/factsheet_netcdf.pdf

Unidata. (2020). *NetCDF: An Introduction to NetCDF*. https://www.unidata.ucar.edu/software/netcdf/docs/netcdf_introduction.html#netcdf_4_format

Wang, L., Yan, J., Ma, Y., Yan, J., & Ma, Y. (2019). Remote Sensing Product Production in an Open-Stack-Based Cloud Computing Environment. In *Cloud Computing in Remote Sensing* (pp. 151–173). Chapman and Hall/CRC. https://doi.org/10.1201/9780429488764-7

Woodcock, R., Cecere, T., Mitchell, A., Killough, B., Dyke, G., Ross, J., Albani, M., Ward, S., & Labahn, S. (2016). *CEOS Future Data Access & Analysis Architecture Study*. http://ceos.org/document_management/Meetings/Plenary/30/Documents/5.2_Future-Data-Architectures-Interim-Report_v.1.pdf

Wu, Y., Xiang, Y., Ge, J., & Muller, P. (2018). High-Performance Computing for Big Data Processing. *Future Generation Computer Systems*, *88*, 693–695. https://doi.org/10.1016/j.future.2018.07.054

xarray Developers. (2020a). *Data Structures*. Xarray 0.15.1 Documentation. https://xarray.pydata.org/en/stable/data-structures.html

xarray Developers. (2020b). *Reading and writing files*. Xarray 0.15.1 Documentation. https://xarray.pydata.org/en/stable/io.html#zarr

Yee, C., Durbin, C., Quinn, P., & Shum, D. (2020). *Task 51—Cloud-Optimized Format Study* (Technical Paper No. EED2-TP-125, Rev. 01). Raytheon Company. http://wiki.esipfed.org/images/3/38/EED2-TP-125_Rev01_CloudOptimizeStudy.pdf

Zarr Developers. (2019, July 12). *Zarr*. Zarr 2.4.0 Documentation. https://zarr.readthedocs.io/en/stable/index.html

Zarr Developers. (2020, January 9). *Tutorial*. Zarr 2.4.0 Documentation. https://zarr.readthedocs.io/en/stable/tutorial.html

Annex A date_interval_endpoints

Source: Own work

```
def date_interval_endpoints(starttime, endtime, day_of_new_interval):
    """
    Return a list of half-month endpoints.

    Keyword arguments:
    - starttime: datetime or date
    - endtime: datetime or date
    - day_of_new_interval: int

    Returns:
    - dates: list(datetime)
    """

    from datetime import datetime
    from dateutil.relativedelta import relativedelta as rdelta
    from dateutil.rrule import rrule, MONTHLY
    from pandas import to_datetime

    starttime = datetime(*starttime.timetuple()[:3],0,0)
    endtime = datetime(*endtime.timetuple()[:3],0,0)
    d=day_of_new_interval

    dates = list(rrule(MONTHLY, dtstart=starttime, until=endtime, bymonthday=[
1,d-1,d,-1]))

    # add starttime/endtime if not included in dates
    if not dates[0].day == 1 and not dates[0].day == d:
        dates = [starttime] + dates

    if (not dates[-1].day == to_datetime(dates[-1]).daysinmonth and
        not dates[-1].day == d-1):
        dates = dates + [endtime]

    # set time of right endpoints to 23:59:59
    for i in range(1,len(dates),2):
        dates[i] = dates[i]+rdelta(hour=23, minute=59, second=59)

    return dates
```

Annex B Batch Processing code

B.1 Evalscript

Source: Milcinski, 2020a; own work

```
//VERSION=3
//double curly brackets render as single curly brackets in python format strin
gs

/* FUNCTIONS */

function setup() {{
  return {{
    input: [{{
      bands: {bands},
      units: "DN"
    }}],
    output: {output_array},
    mosaicking: Mosaicking.ORBIT
  }}
}}

function validate (sample) {{
  if (sample.dataMask!=1) return false;

  var scl = sample.SCL//Math.round(sample.SCL);

  if (scl === 3) {{ // SC_CLOUD_SHADOW
    return false;
  }} else if (scl === 9) {{ // SC_CLOUD_HIGH_PROBA
    return false;
  }} else if (scl === 8) {{ // SC_CLOUD_MEDIUM_PROBA
    return false;
  }} else if (scl === 7) {{ // SC_CLOUD_LOW_PROBA
    //return false;
  }} else if (scl === 10) {{ // SC_THIN_CIRRUS
    return false;
  }} else if (scl === 11) {{ // SC_SNOW_ICE
    return false;
  }} else if (scl === 1) {{ // SC_SATURATED_DEFECTIVE
    return false;
  }} else if (scl === 2) {{ // SC_DARK_FEATURE_SHADOW
    //return false;
  }}
  return true;
}}

function calculateIndex(a,b)
{{
  if ((a+b)==0) return 0;
  // stretch [-1,+1] to [0,1]
  return ((a-b)/(a+b)+1)/2;
}}

function interpolatedValue(arr)
{{
  //here we define the function on how to define the proper value -
 e.g. linear interpolation; we will use average
  if (arr.length==0) return 0;
  if (arr.length==1) return arr[0];
  var sum = 0;
  for (j=0;j<arr.length;j++)
  {{sum+=arr[j];}}
  return Math.round(sum/arr.length);
}}

function fillResultArray(i, int_bands)
{{
  for (var k=0; k<bands.length; k++) {{
    if(int_bands[bands[k]].length==0) results[bands[k]][i] = 0
    else results[bands[k]][i] = interpolatedValue(int_bands[bands[k]])
  }}

  for (var k=0; k<ixs.length; k++) {{
    if(ixs[k]!=="CVI") {{
        results[ixs[k]][i] = 65535*calculateIndex(
            results[ic[ixs[k]][0]][i],
            results[ic[ixs[k]][1]][i]
        )
    }} else {{
        // output sample type for CVI is FLOAT32
        results[ixs[k]][i] = results["B08"][i]*results["B05"][i] / (results["B
03"][i]*results["B03"][i])
    }}
  }}
}}


/* MAIN */

var ic = {{  // index components
  "NDVI":  ["B08", "B04"],
  "GNDVI": ["B08", "B03"],
  "BNDVI": ["B08", "B02"],
  "NDSI":  ["B11", "B12"],
  "NDWI":  ["B03", "B08"]
}}
```

```
var bands = Object.keys({int_bands})
var ixs = {indices}

var results = {results_object}

// We split each month into two halves. This will make it easier to append mon
ths to data cube later
var day_of_new_interval = {day_of_new_interval}
var endtime = new Date({enddate_unix}) // UNIX epoch in ms

function evaluatePixel(samples, scenes) {{

  var is_in_last_half_of_month = endtime.getUTCDate() >= day_of_new_interval
  var i = 0; // interval number
  var int_bands_empty = {int_bands}
  var int_bands = int_bands_empty

  for (var j = 0; j < samples.length; j++) {{

    // if scene is outside of current half of month, fill result array and cha
nge half of month
    // algorithm starts with most recent observation
    if (( !is_in_last_half_of_month && scenes[j].date.getUTCDate() >= day_of_n
ew_interval) ||
    (  is_in_last_half_of_month && scenes[j].date.getUTCDate() <  day_of_new_i
nterval))
    {{
      fillResultArray(i, int_bands)

      int_bands = int_bands_empty //reset values
      is_in_last_half_of_month = !is_in_last_half_of_month;
      i++;
    }}

    if (validate(samples[j]))
    {{
      // push input samples into their respective arrays
      for (var k=0; k<bands.length; k++) {{
        int_bands[bands[k]].push(samples[j][bands[k]])
      }}
    }}
  }}

  //execute this for the last interval
  fillResultArray(i, int_bands);

  return results
}}
```

B.2 EDC Batch Processing Jupyter Notebook

```
# load credentials from environment variables
%load_ext dotenv
%dotenv

# util
import boto3

# date & time
from datetime import date, datetime
from util import date_interval_endpoints as endpoints

# Oauth
from oauthlib.oauth2 import BackendApplicationClient
from requests_oauthlib import OAuth2Session
```

B.2.1 Get authorization token

Source: Sentinel Hub, 2020a; own work

```
# Your client credentials
client_id = %env SH_CLIENT_ID
client_secret = %env SH_CLIENT_SECRET

# Create a session
client = BackendApplicationClient(client_id=client_id)
oauth = OAuth2Session(client=client)

token = oauth.fetch_token(token_url='https://services.sentinel-
hub.com/oauth/token',
                          client_id=client_id, client_secret=client_secret)
```

B.2.2 Configure request (evalscript)

Enter start and end date, input bands, indices. The resulting files will have two time intervals per month, being split at *day_of_new_interval*.

```
startdate = date(2017,9,1) # Y,M,D
enddate = date(2018,05,15)  # Y,M,D

input_bands = [
    "B02",
    "B03",
    "B04",
    "B05",
    "B06",
    "B07",
    "B08",
    "B8A",
    "B11",
    "B12"
]


indices = [
    "NDVI",
    "GNDVI",
    "BNDVI",
    "CVI",
    "NDSI",
    "NDWI"
]

bucket_name = "eox-masterdatacube"

day_of_new_interval = 16 # leave this unchanged in most of the cases
```

B.2.3 Calculate parameters

```
eps = endpoints(startdate, enddate, day_of_new_interval)

timestamps  = [int(d.timestamp()) for d in eps]   # timestamps for arithmetic
avg_times = [(left+right)/2 for left,right in zip(timestamps[::2],timestamps[1
::2])]
avg_times = [datetime.utcfromtimestamp(a) for a in avg_times]
avg_times = [dt.isoformat() for dt in avg_times]
```

```
masks = ["SCL", "dataMask"] # SCL ... Scene Classification Layer

output_bands = input_bands + indices
output_array =  [ { 'id': "\"" + ob + "\"", 'bands': len(avg_times), "sampleTy
pe": "SampleType.UINT16"} for ob in output_bands ]
for oa in output_array:
    if oa["id"] == '"CVI"':
        oa["sampleType"] = "SampleType.FLOAT32"
output_array = str(output_array).replace("'", '')

int_bands = '{' + ','.join([f'{ib}: []' for ib in input_bands]) + '}'
results_object = '{' + ','.join([f'{ob}: []' for ob in output_bands]) + '}'
responses = [{"identifier": ob,"format": {"type": "image/tiff"}} for ob in out
put_bands]
```

B.2.4 Evalscript & Payload

Source: Sentinel Hub, 2020e; own work

```
with open('evalscript.js', 'r') as file:
    evalscript = file.read()

evalscript = evalscript.format(
    bands               =str(input_bands+masks),
    output_array        =output_array,
    results_object      =results_object,
    day_of_new_interval=day_of_new_interval,
    enddate_unix        =datetime(*enddate.timetuple()[:3],23,59,59).timestamp(
)*1000,
    int_bands           =int_bands,
    indices             =indices
)
```

```python
mab_geometry = { "type": "Polygon", "coordinates":
    [
        [
            [ 14.249086675661697, 48.549686576488739, 0.0 ],
            [ 14.804694846753847, 48.546347802635076, 0.0 ],
            [ 14.804740209647099, 48.662420784507489, 0.0 ],
            [ 14.802209327231056, 48.75281114695149, 0.0 ],
            [ 16.224291241591107, 48.74506103134155, 0.0 ],
            [ 16.19780424032631, 47.621347235969026, 0.0 ],
            [ 14.263435524016538, 47.627863161515464, 0.0 ],
            [ 14.249086675661697, 48.549686576488739, 0.0 ]
        ]
    ]
}

payload = {
  "processRequest": {
    "input": {
      "bounds": {
        "properties": {
          "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        },
        "geometry": mab_geometry
      },
      "data": [
        {
          "type": "S2L2A",
          "dataFilter": {
            "timeRange": {
              "from": eps[0].isoformat() + 'Z', # starttime
              "to": eps[-1].isoformat() + 'Z'   # endtime
            },
            "mosaickingOrder": "mostRecent",
            "maxCloudCoverage": 100,
            "previewMode": "DETAIL"
          }
        }
      ]
    },
    "output": {
      "responses": responses
    },
    "evalscript": evalscript
  },
  "tilingGridId": 0,
  "bucketName": bucket_name,
  "resolution": 10.0,
  "description": "Test MAB"
}
```

B.2.5 Send request

Source: Sentinel Hub, 2020e; own work

```
url = 'https://services.sentinel-hub.com/batch/v1/process/'

response = oauth.request("POST", url, json = payload).json()
request_id = response["id"]
print(f"Status of request {request_id}: {response['status']}")

oauth.request("POST", f'{url}{request_id}/start')
print('Processing started.')

s3 = boto3.resource('s3')
bk = s3.Bucket(bucket_name)

bk.put_object(Key=request_id + '/userdata.json', Body=json.dumps({
    'bands': output_bands,
    'request_id': request_id,
    'time': avg_times
}))
print('Metadata saved to bucket')
```

Annex C mapchete Hub code

C.1 datacube.py

Source: Ungar, 2020b; own work

```
from datetime import datetime
import numpy as np
from shapely.geometry import shape
from pandas import IntervalIndex
import xarray as xr

from mapchete_satellite.exceptions import EmptyStackException
from mapchete_satellite.settings import SENTINEL2_BAND_INDEXES

def execute(
    mp,
    bands=[2, 3, 4, 5, 6, 7, 8, 9, 11, 12],
    resampling="cubic_spline",
    read_threads=1
):
    """
    Extract satellite data slices to 4D xarray.

    Inputs
    ------
    satellite_cube
        S2AWS or S2Mundi input

    Parameters
    ----------
    bands : int or list of int
        Indexes of bands considered.
    resampling : str (default: 'nearest')
        Resampling used when reading data.
    read_threads : 1
        Number of parallel read threads.

    Output
    ------
    xarray.DataArray
    """
    if "aoi" in mp.params["input"]:
        with mp.open("aoi") as aoi:
            if not len(aoi.read()):
                return "empty"

    with mp.open("satellite_cube") as sat:
        try:
            # create 4D xr.DataArray with named slice_ids and named bands
            in_cube = sat.read_cube(
                indexes=bands,
                resampling=resampling,
                mask_clouds=True,
                threads=read_threads
            )
            band_names = [SENTINEL2_BAND_INDEXES[sat.processing_level][i] for
i in bands]

            cube = xr.DataArray(
                # apply masks and swap "bands" and "timestamp" axes
                in_cube.data.transpose(1, 0, 2, 3),
                # named dimension indexes
                coords={
                    "bands": [b.split('_')[0] for b in band_names],
                    "timestamps": list(in_cube.timestamps),
                },
                # named dimensions
                dims=("bands", "timestamps", "x", "y"),
            )

            # temporarily convert to xarray.DataSet
            cube = cube.to_dataset("bands")

            # new interval starts at day 16 of month
            eps = date_interval_endpoints(*sat._time_range, 16)
            int_idx = IntervalIndex.from_arrays(eps[::2], eps[1::2])
            avg_cube = cube.groupby_bins('timestamps', bins=int_idx).mean('tim
estamps')
            avg_cube = avg_cube.rename({'timestamps_bins': 'time'}) # xcube Da
taset spec
            avg_cube.coords['time'] = int_idx.mid # zarr cannot have IntervalI
ndex as coords

            for idx, (ic1, ic2) in ics.items():
                if ic1 in avg_cube and ic2 in avg_cube:
                    # 2**16
                    avg_cube[idx] = calculate_index(avg_cube[ic1], avg_cube[ic
2]) * 65335

            # CVI calculation
            if "B03" in avg_cube and "B05" in avg_cube and "B08" in avg_cube:
                avg_cube['CVI'] = (avg_cube.B08 * avg_cube.B05 / avg_cube.B03*
*2) * 1000

            # Typing conforming to SH Mass output
            avg_cube = avg_cube.astype(np.uint16)

            # convert to xarray.DataArray again for writing
            return avg_cube.to_array("bands")

        except EmptyStackException:
            return "empty"


def date_interval_endpoints(starttime, endtime, day_of_new_interval):
...

def calculate_index(a, b):
    """Calculate one of the ices indexes."""
    # stretch [-1,+1] to [0,1]
    return ((a - b) / (a + b) + 1) / 2

# index components
ics = {
    "NDVI": ["B08", "B04"],
    "GNDVI": ["B08", "B03"],
    "BNDVI": ["B08", "B02"],
    "NDSI": ["B11", "B12"],
    "NDWI": ["B03", "B08"]
}
```

C.2 datacubes.mapchete

Source: Ungar, 2020a; own work

```yaml
process: datacube.py
input:
  aoi: s3://eox-masterdatacube/mapchete_cubes/mab.geojson
  satellite_cube:
    format: S2AWS
    level: L2A
    with_cloudmasks: true
    start_time: 2019-09-01
    end_time: 2020-05-15
    max_products: 3700
    remote_timeout: 60
    cache:
      path: /mnt/data/cache
      intersection_percent: 0.5
      bands: [2, 3, 4, 5, 6, 7, 8, 9, 11, 12]
output:
  format: xarray
  path: s3://eox-masterdatacube/mapchete_cubes/mdc_17_01/
  dtype: uint16
  storage: zarr
pyramid:
  grid:
    shape: [115, 9]
    bounds: [186210.0, -9800.0, 923490.0, 9411000.0]
    is_global: False
    epsg: 32633
  metatiling: 2
zoom_levels: 5
```