eodc Earth Observation Data Centre

Cloud Service Lösungen

für Forschende und Studierende der Fernerkundung und Photogrammmetrie

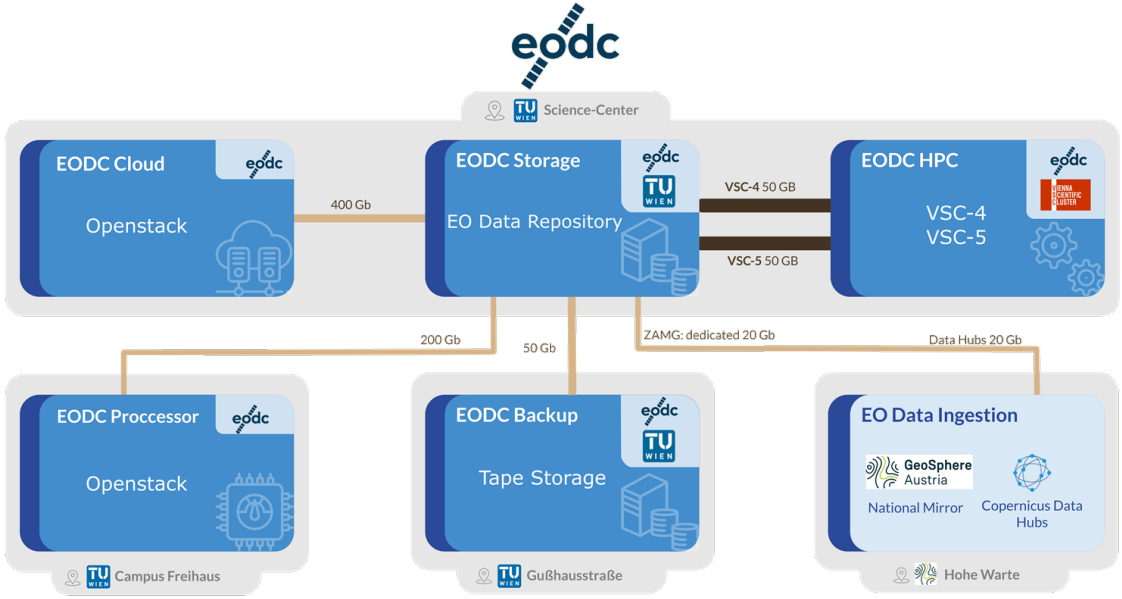
Impulsvortrag: EODC – Christoph Reimer

Wer sind wir?

- Gründung am 15.05.2014 einer Public-Private Partnership
- Gesellschafter:
  - Public: Technische Universität, GeoSphere Austria
  - Private: Cloudflight Austria GmbH, GeoVille GmbH,
    3 Privatpersonen
- Ziele der Public-Private Partnership
  - Stärkung der Zusammenarbeit zwischen Öffentlichen und Privaten Sektor
  - Etablierung einer gemeinsamen IT-Infrastruktur
  - Brücke zwischen Wissenschaft und Anwendung

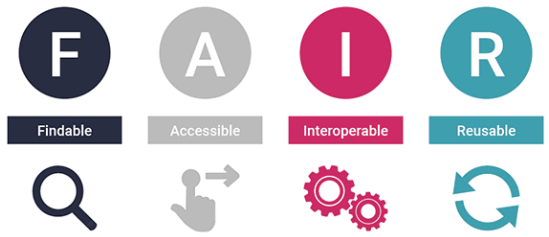
Was machen wir?

- Vision: “translating data into knowledge”
- Wir unterstützen unsere Kunden mit
  - maßgeschneiderten Cloud Services für Anwendungen in der Fernerkundung und Photogrammetrie
  - IT Infrastruktur [Storage: 18 PB Disk, 40PB Tape]
  - Daten Management
- Services für ESA/EC/EUMETSAT
  - Real-Time Daten Services für End-User



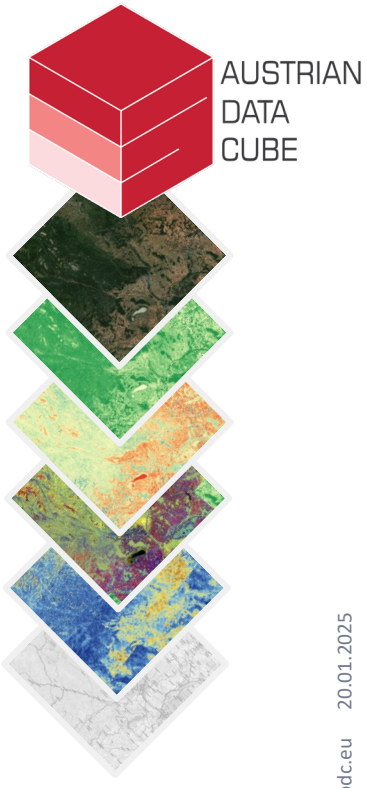
Warum machen wir das?

- User sollen schnell und einfach gewünschte Daten finden
- Daten sollen für User einfach zugänglich sein
- Daten Formate für Metadaten und Daten sollen selbsterklärend und einfach zu verstehen sein.



Wie ermöglichen wir diesen Zugriff?

- Einsatz von Technologien zur Realisierung von “Data Cubes”
- Austrian Data Cube als erstes “Proof of Concept”
  - Daten werden via Raum-Zeit Abfrage direkt als Object (DataCube) geladen [Python]
  - Zusammenführen von Metadaten und Daten (Index)
- Community Standard STAC [Metadaten]
  - Dient zur Suche und Erkundung von Geodaten
  - fokussiert sich ausschließlich auf Raster/Array Daten
  - STAC API stellt eine Anzahl an HTTP Endpunkten zur Verfügung um nach bestimmten “STAC Items” suchen zu können.
- Einsatz von “Cloud native” Daten Formaten

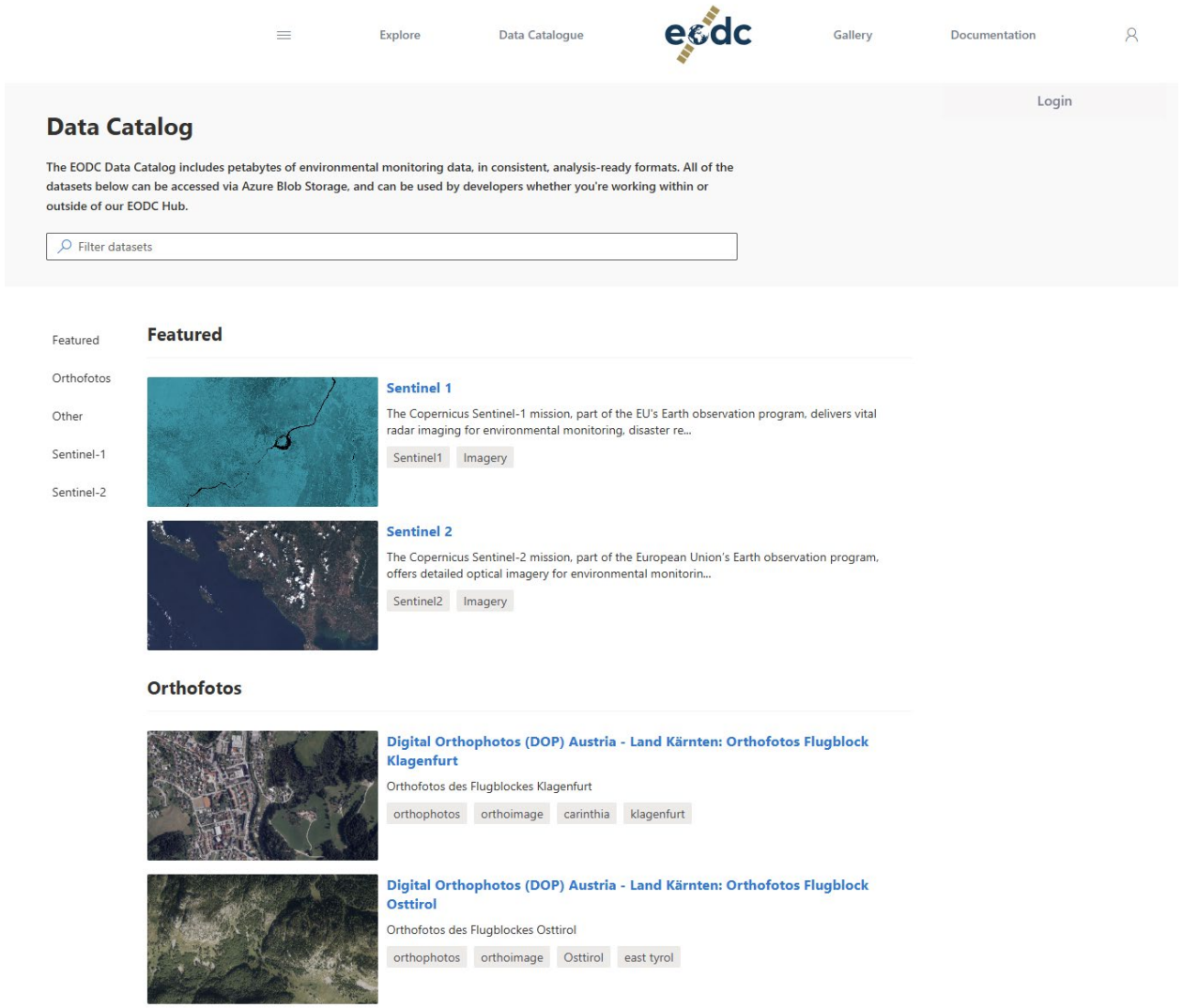


Daten APIs – Cloud Native Data Formats

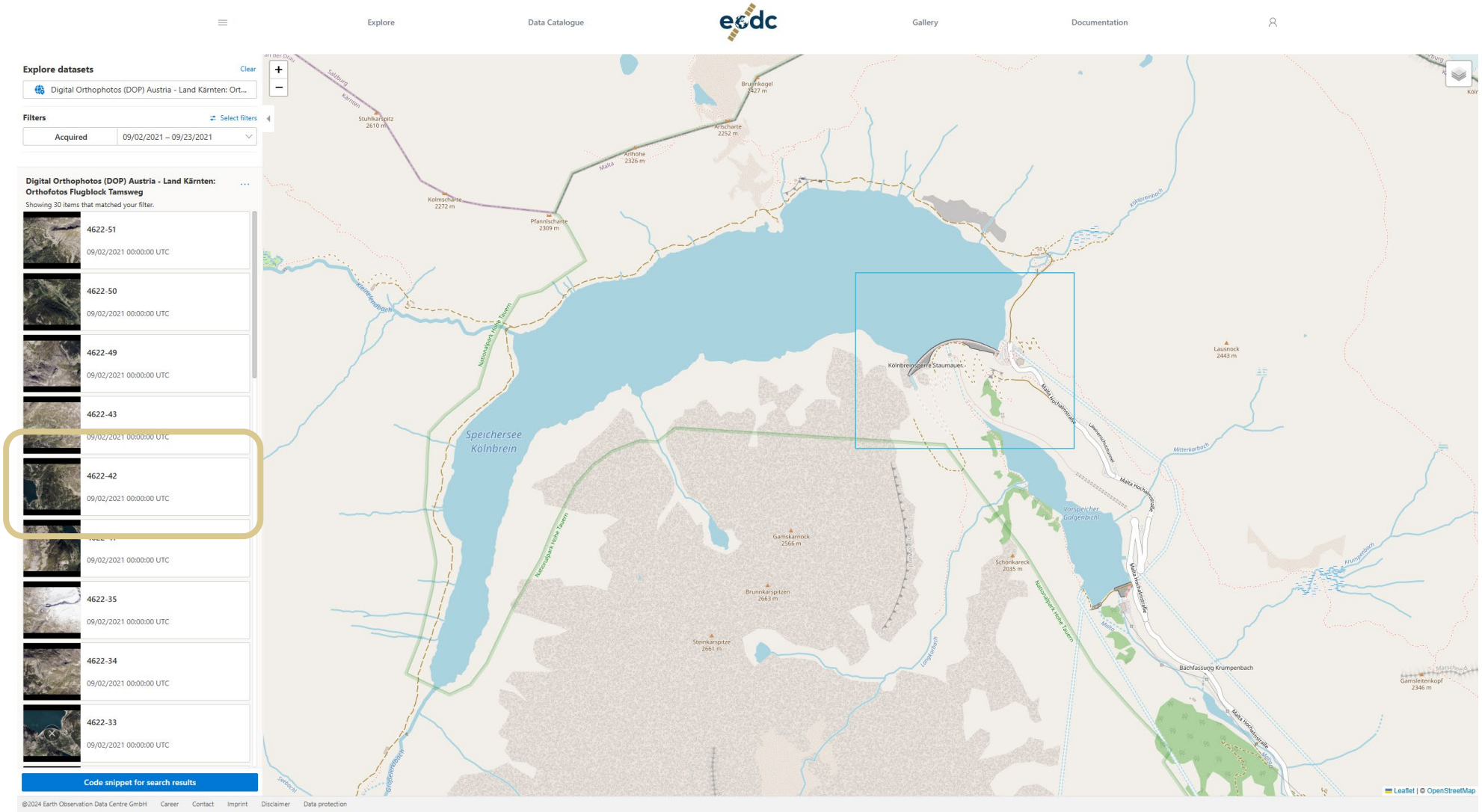
- Einfacher und schneller Zugriff auf Daten via HTTP
  - kein kompliziertes navigieren auf einem Filesystem
- Unterschiedliche HTTP APIs für Zugriff auf Daten:
  - eodc dataAPI: read-only Zugriff auf File basierten Storage
  - S3 API: Zugriff auf Cloud Object Storage
- Zusätzliche Funktionen verfügbar (eodc dataAPI)
  - Granulare Zugriffskontrolle (RBAC, PBAC)
    - Teilen mit anderen Usern
  - Direkter Zugriff auf individuelle Files in einem ZIP Archiv, etc.
  - Lesen von Teilen einer Datei via HTTP Range-Requests

COG CLOUD OPTIMIZED GEOTIFF Zarr [tile]DB GeoJSON Parquet

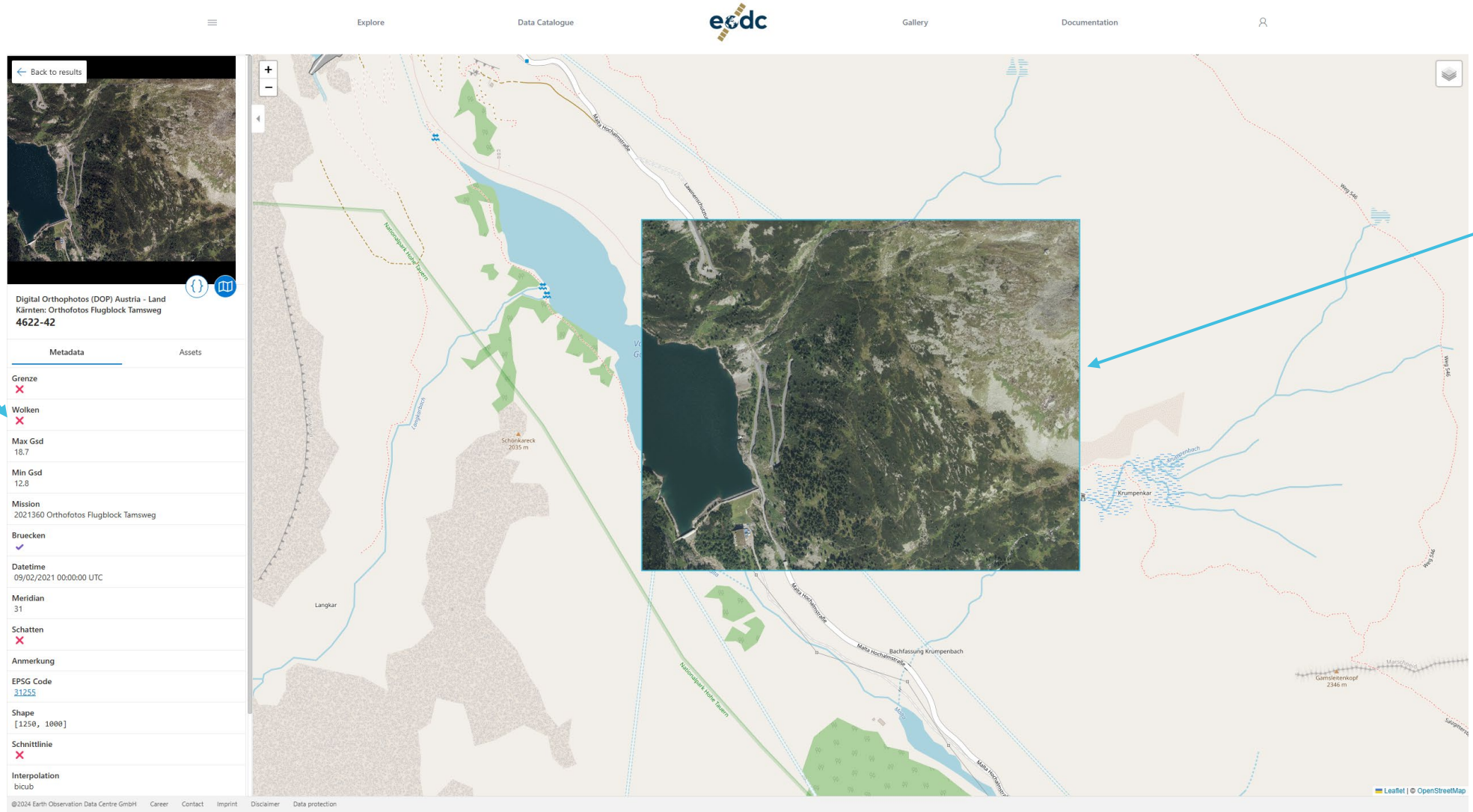
Was kann man damit nun tun?



Visualisieren der Daten



Visualisieren der Daten



Metadaten

Daten

Analyse der Daten in Python

- Python Bibliotheken für direkten Zugriff auf STAC API und Daten APIs

```python
# Necessary python imports

from pystac_client import Client
from odc import stac as odc_stac
import pyproj
import xarray as xr
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
```

Analyse der Daten in Python

```python
# Search against the EODC STAC API
catalog = Client.open(
  "https://stac.eodc.eu/api/v1"
)

# Define your area of interest
aoi = {
  "type": "Polygon",
  "coordinates": [
    [
      [16.432800292968754, 50.747318126029434],
      [17.87887573242188, 50.747318126029434],
      [17.87887573242188, 51.306150195330034],
      [16.432800292968754, 51.306150195330034],
      [16.432800292968754, 50.747318126029434]
    ]
  ]
}

# Define your temporal range
daterange = {"interval": ["2024-09-18T00:00:00Z", "2024-09-25T00:00:00Z"]}

# Define your search with CQL2 syntax
search = catalog.search(filter_lang="cql2-json", filter={
  "op": "and",
  "args": [
    {"op": "s_intersects", "args": [{"property": "geometry"}, aoi]},
    {"op": "anyinteracts", "args": [{"property": "datetime"}, daterange]},
    {"op": "=", "args": [{"property": "collection"}, "GFM"]}
  ]
})

# Retrieve all found items
items = search.item_collection()
print("We found", len(items), "items, that match our filter criteria.")
```

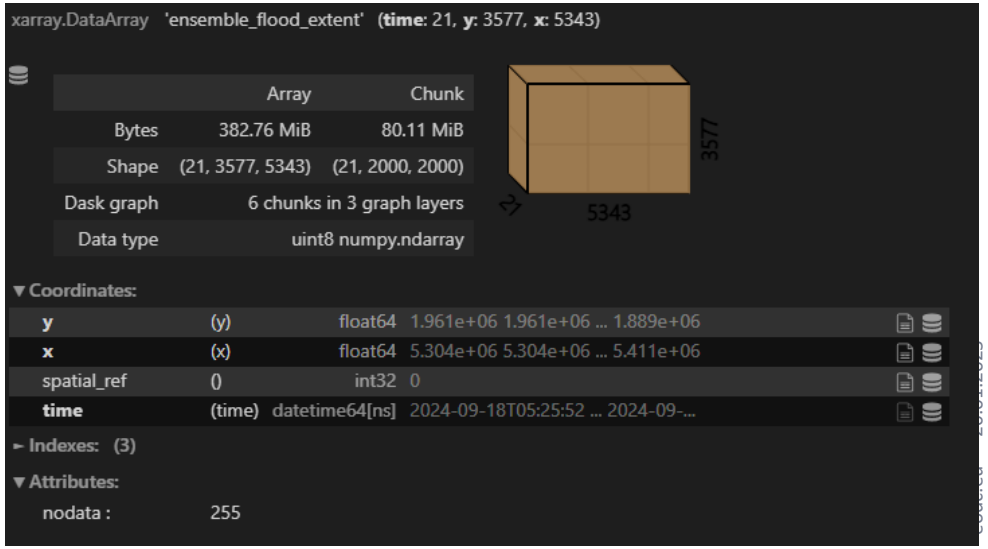
Analyse der Daten in Python

```python
# Derive Equi7Grid CRS from first found item
crs = pyproj.CRS.from_wkt(items[0].properties["proj:wkt2"])

# Define assets to load
assets = ["ensemble_flood_extent"]

# Get bounding box from AOI
polygon = Polygon(aoi['coordinates'][0])

# Load asset data into xarray using odc-stac
# Adjust chunk size of x/y according to available RAM
xx = odc_stac.load(
    items,
    bbox=polygon.bounds,
    crs=crs,
    bands=assets,
    dtype="uint8",
    chunks={"x": 2000, "y": 2000, "time": -1},
    resolution=20)
```

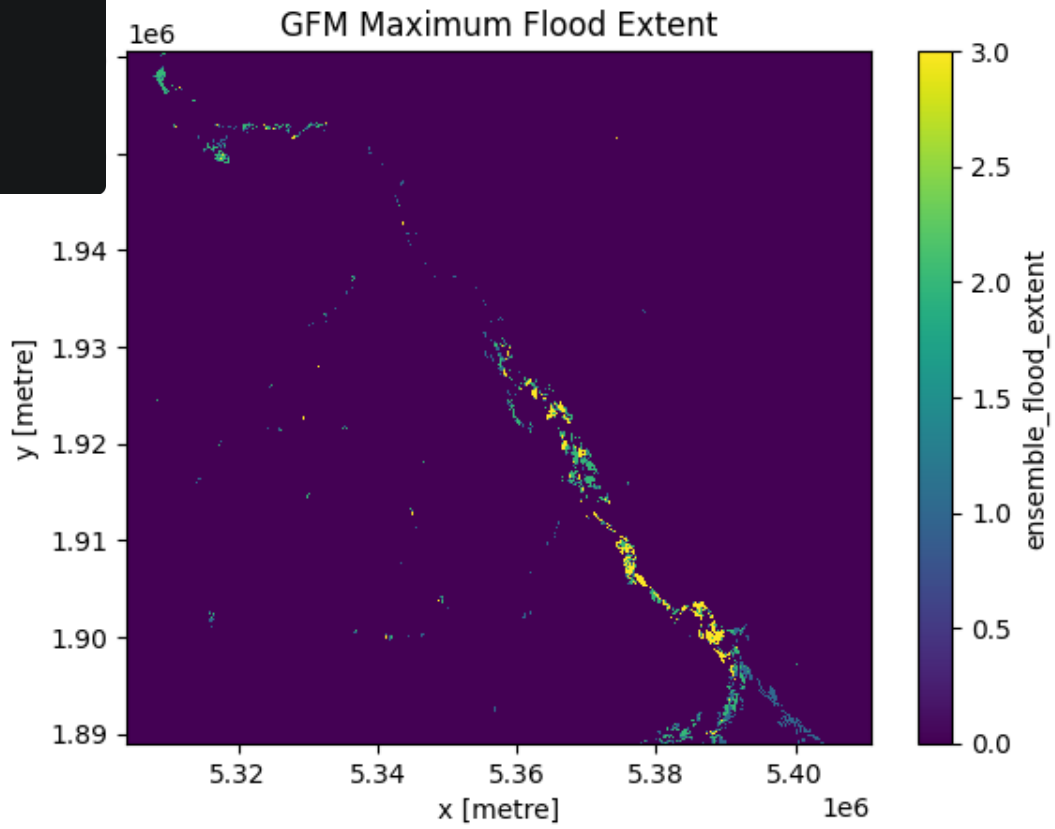


Analyse der Daten in Python

```python
# Mask data which is nodata (255) and non-flood (0)
xx = xx.where((xx != 255) & (xx != 0))

# Compute sum along time dimension
data = xx.sum(dim="time").astype("uint8")
data.ensemble_flood_extent

data.ensemble_flood_extent.plot()
plt.title("GFM Maximum Flood Extent")
plt.show()
```

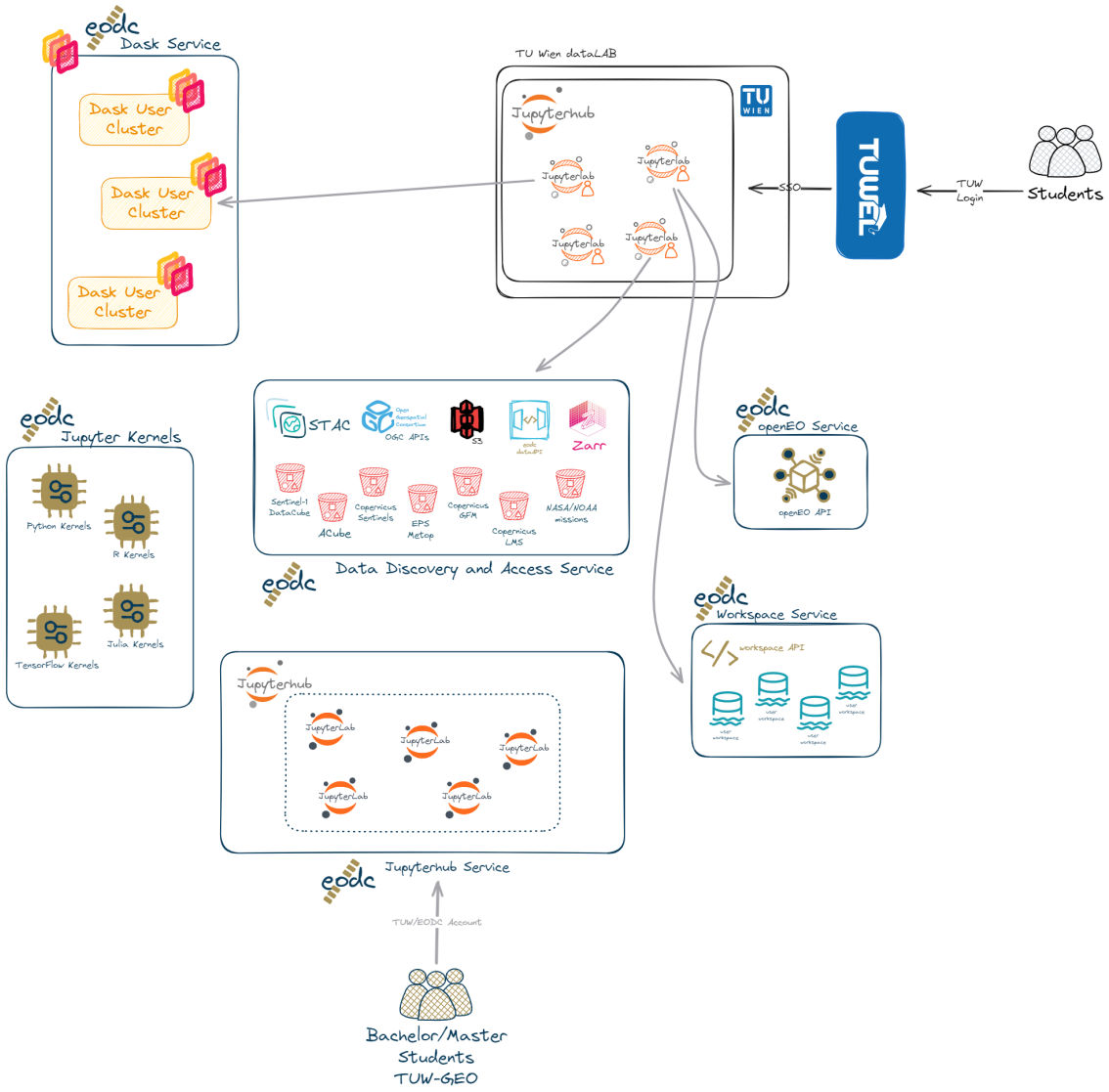


Datenanalyse in der Cloud für Forscher

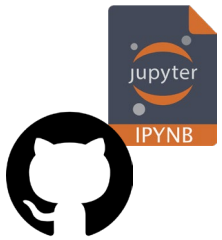
- Wie kann ich meine Software/Code in einer Cloud Umgebung ausführen?
- IaaS Ansatz: Virtuelle Maschine (VM)
  - Oft zu kompliziert und skaliert nicht
- Jupyter Ecosystem: JupyterLab, Jupyter Notebook
- Dask: Python Framework für “Parallel and distributed computing”
  - Ausführen von Python Code in einer Cloud Umgebung

Cloud Service Lösung

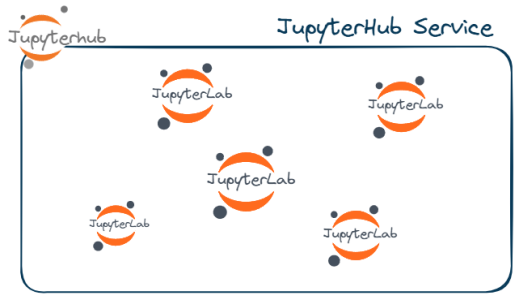
- Daten müssen nicht kopiert und für Aufgaben breitgestellt werden.
- Keine Aufbereitung der Daten für Übungen notwendig
- “Data proximate computing” via Dask
  - Skaliertes Rechnen nahe der Daten (on-demand)
  - Es werden nur finale Resultate übertragen



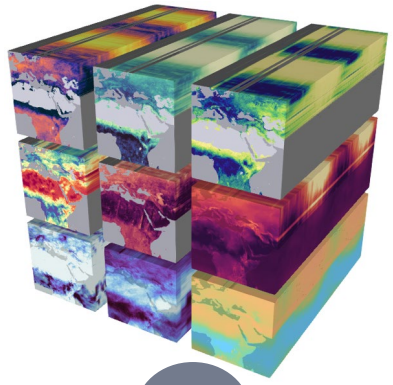
Dask in Anwendung

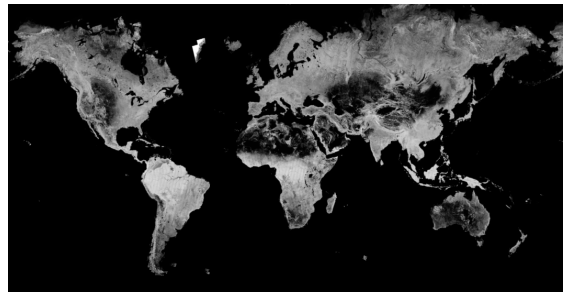


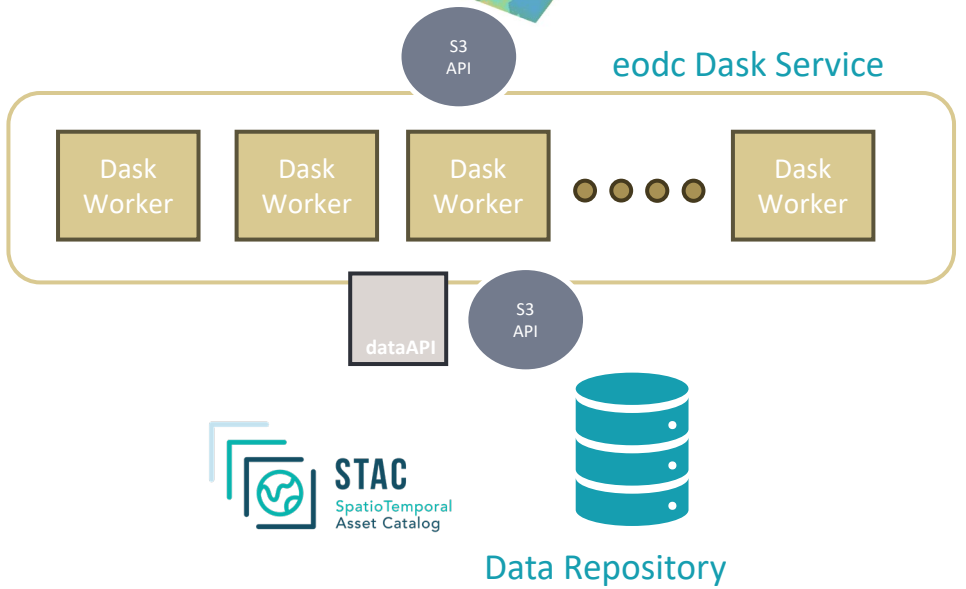
User Interface

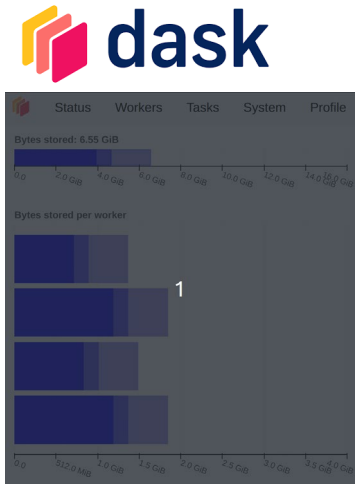


http://









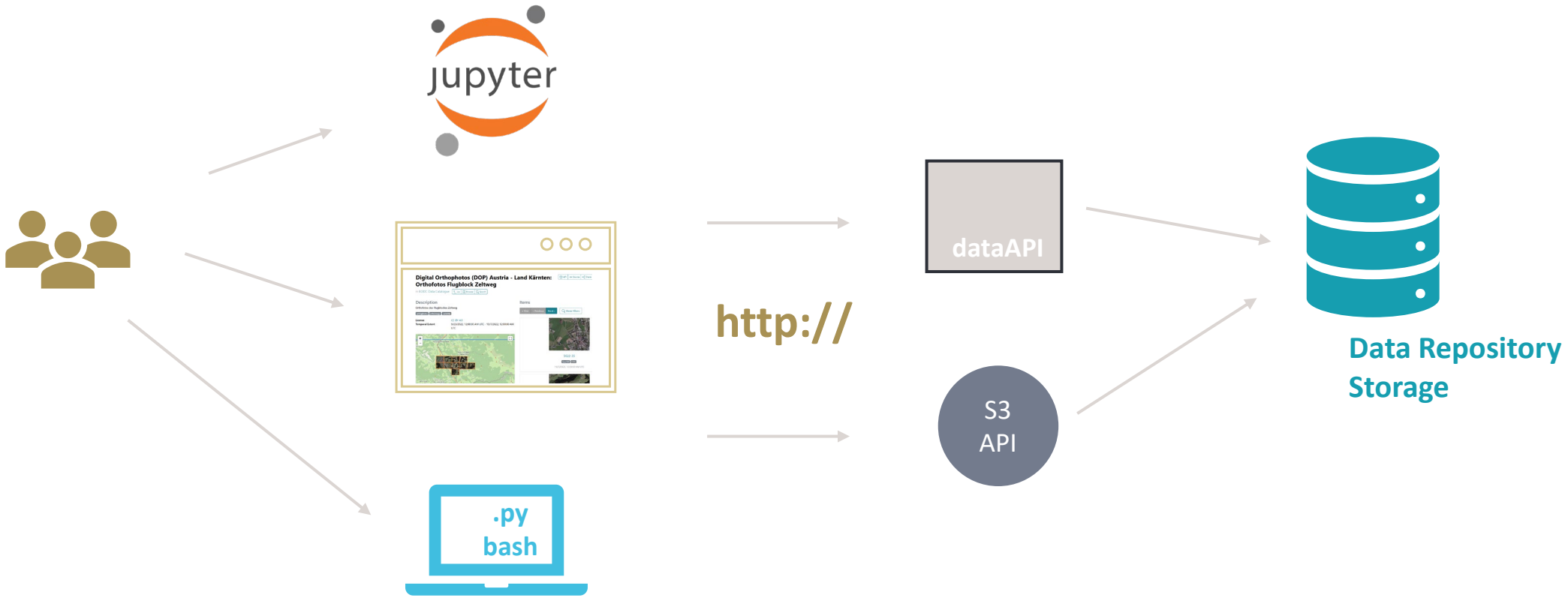
eodc Earth Observation Data Centre

Danke

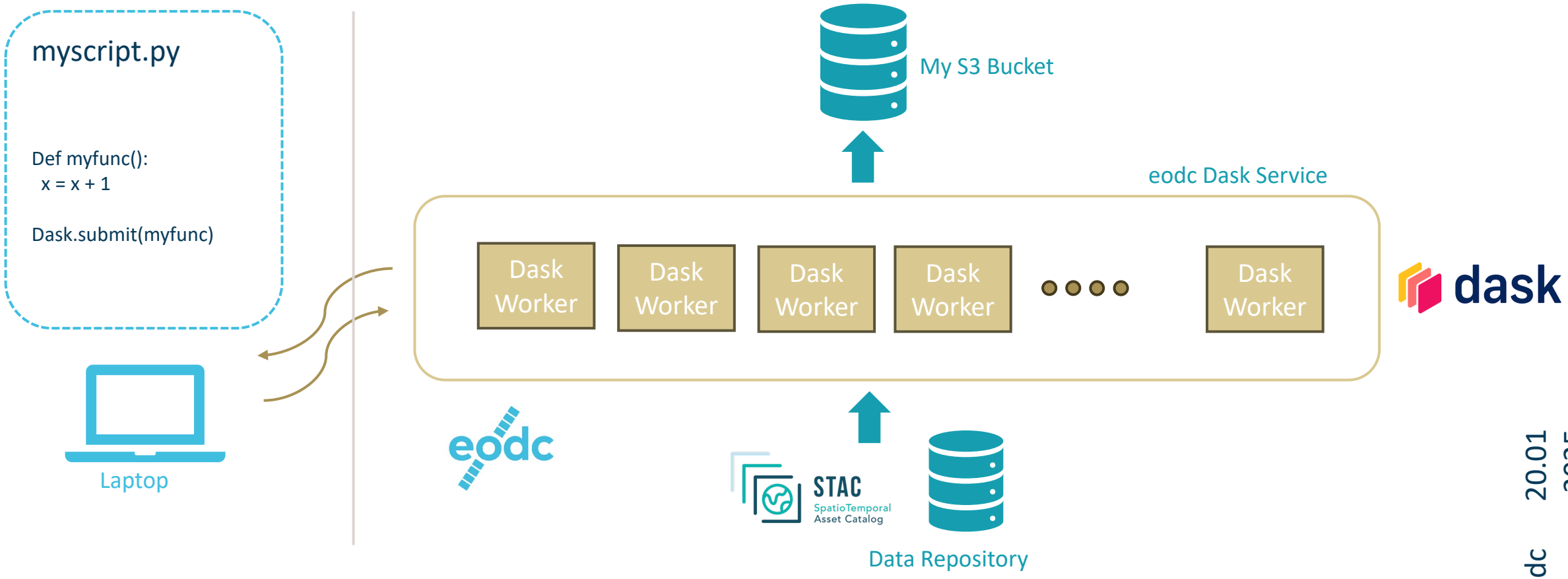
Details zu den einzelnen Services am eodc Stand


eodc
Translating data
into knowledge

Daten APIs –Cloud Native



Anwendung von Dask



ArgoWorkflows Use Case

myargowf.py

```python
from hera.workflows import DAG, Workflow, script


@script()
def echo(message):
    print(message)


with Workflow(generate_name="dag-diamond-", entrypoint="diamond") as w:
    with DAG(name="diamond"):
        A = echo(name="A", arguments={"message": "A"})
        B = echo(name="B", arguments={"message": "B"})
        C = echo(name="C", arguments={"message": "C"})
        D = echo(name="D", arguments={"message": "D"})

        A >> [B, C] >> D
```

Laptop